DATA MANAGEMENT IN AN OBJECT-ORIENTED DISTRIBUTED AIRCRAFT CONCEPTUAL DESIGN ENVIRONMENT

A Thesis
Presented to
The Academic Faculty

By

Zhijie Lu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2007

DATA MANAGEMENT IN AN OBJECT-ORIENTED DISTRIBUTED AIRCRAFT CONCEPTUAL DESIGN ENVIRONMENT

Approved by:

Dr. Dimitri Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Daniel Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. James Craig
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Neil Weston
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Russell Peak
Manufacturing Reseach Center
Georgia Institute of Technology

Date Approved: January 2007

Dare to dream.

ACKNOWLEDGEMENTS

First and foremost, I sincerely thank my advisor, Dr. Dimitri Mavris, for his tremendous guidance and support, for having me in the ASDL family, and most of all, for giving me the opportunity to learn, to grow, and to explore. Not only do I respect him as my research advisor, I also look up to him as my mentor for my career life.

I thank my committee members, Dr. Daniel Schrage, Dr. James Craig, Dr. Russell Peak, and Dr. Neil Weston for their precious time, their insightful comments, suggestions, and encouragements. Especially, I thank Dr. Neil Weston for helping me see through the mist when I lost direction.

I would also like to express my gratitude to Dr. Byung-Ho Ahn for providing me the decomposed ACSYNT disciplinary analyses components. I thank Henry Won for teaching me how a hybrid propulsion system works, and for the *Matlab* codes. Thanks are also due to Robert McDonald and Cara Zell, who took the time to review my thesis editorially. Many thanks to Eunsuk Yang and Cara Zell for the encouragements they gave me when I needed them the most.

Always, I thank my family, for their love, and for giving me the courage and strength of working through the challenges, and, to carry on.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1NF	The First Normal Form
2NF	The Second Normal Form
3NF	The Third Normal Form
ACSYNT	Aircraft Synthesis Program
AEE	Advanced Engineering Environment
AML	Adaptive Modeling Language
API	Application Programmer Interface
ASDL	Aerospace Systems Design Laboratory
ASSET	Advanced Surface Ship Evaluation Tool
BFGS	The Broyden-Fletcher-Goldfarb-Shano Algorithm
BWB	Blend Wing Body
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CASDAT	The Conceptual Aerospace System Design and Analysis Toolkit
CDA	Contributing Disciplinary Analysis
CDATA	Character Data
CE	Concurrent Engineering
CFD	Computational Fluid Dynamic
CLI	Command Line Interface
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance Domain
CODA	Collaboration Oriented Data Agent
DBMS	Database Management System
DoD	The Department of Defense
DOE	Design of Experiment
DOM	Document Object Model
DSM	Design Structure Matrix
DTD	Document Type Definition
EER	The Enhanced Entity-Relational Model
ER	The Entity-Relational Model

FIPER	Federated Intelligent Product Environment
FLOPS	Flight Optimization System
FNC	Federal Networking Council
GASP	General Aviation Synthesis Program
GTPDP	Georgia Tech Preliminary Design Program
GUI	Graphical User Interface
HAVOC	Hypersonic Aircraft Vehicle Optimization Code
HESCOMP	Helicopter Sizing & Performance Computer Program
HTML	Hyper Text Mark-up Language
HTTP	Hyper Text Transfer Protocol
IDAS	Integrated Design & Analysis System
IDEF	Integrated Definition Methods
IEEE	The Institute of Electrical and Electronics Engineers
IMAGE	The Intelligent Multidisciplinary Aircraft Generation Environment
INCOSE	The International Council on Systems Engineering
IP	Internet Protocol
IPPD	Integrated Product and Process Design
JDBC	Java Database Connectivity
JDO	Java Data Object
JPL	NASA's Jet Propulsion Laboratory
JSF	Joint Strike Fighter
KS	The Kreisselmier-Steinhauser Function
LEAPS	The Leading Edge Advanced Prototyping for Ships
LEGEND	A Laboratory Environment for the Generation, Evaluation, and Navigating of Design
LISI	Levels of Information System Interoperability
LVL	The Launch Vehicle Language
MAO/O	Multidisciplinary Design Analysis/Optimization
MRA	Multi-Representation Architecture
MVC	Model View Controller
NASA	National Aeronautics and Space Administration
NextADE	The Next Generation Aircraft Conceptual Design Software Environment
NPSS	The Numerical Propulsion Simulation System
NSWC	Naval Surface Warfare Center
OAD	Object-Oriented Aircraft Conceptual Design

ODL	Object Definition Language
ODMG	The Object Data Management Group
OKBC	Open Knowledge Based Connectivity
OMG	Object Management Group
OODBS	Object-Oriented Database Management System
OQL	Object Query Language
ORDBMS	Object Relational Database Management System
PBAM	Product Model-Based Analysis Models
PCDATA	Parsed Character Data
PDM	Product Data Management
PHIGS	Programmer's Hierarchical Interactive Graphics System
PSM	Problem Solving Method
QBE	Query by Example
RDS	Robust Design Simulation
RDT&E	Research, Development, Test, and Evaluation
RFP	Request for Proposal
RMI	Remote Method Invocation
RSE	Response Surface Equation
RSM	Response Surface Method
SAX	Simple API for XML
SQL	Structured Query Language
STEP	The Standard for the Exchange of Product Model Data
SysML	Systems Modeling Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
UUID	Universal Unique Identification
VASCOMP	V/STOL Aircraft Sizing & Performance Computer Program
W3C	The World Wide Web Consortium
WWW	The World Wide Web
XML	Extensible Markup Language

NOMENCLATURE

α	-	installed full throttle thrust lapse
β	-	$\frac{W}{W_{TO}}$ and W is the instantaneous weight of the aircraft
C_D	-	drag coefficient
C_L	-	lift coefficient
D	-	drag
g_0	-	acceleration of gravity
h	-	altitude
$K1$	-	coefficient in drag polar ($C_L = K_1 C_L{}^2 + K_2 C_L + C_{D0}$)
$K2$	-	coefficient in drag polar
N	-	load factor
Q	-	dynamic pressure ($= \frac{1}{2}\rho V^2$)
S	-	wing area
ΔS	-	range increment
Δt	-	time increment
T	-	installed thrust
T_{SL}	-	installed thrust at sea level
$TSFC$	-	thrust specific fuel consumption
V	-	speed
W	-	aircraft weight
W_{final}	-	final weight of an aircraft after a mission segment
$W_{initial}$	-	initial weight of an aircraft at the beginning of a mission segment
W_{TO}	-	takeoff gross weight
z_e	-	energy height

SUMMARY

In the competitive global market place, aerospace companies are forced to deliver the right products to the right market, with the right cost, and at the right time. However, the rapid development of technologies and new business opportunities, such as mergers, acquisitions, supply chain management, etc., have dramatically increased the complexity of designing an aircraft. Therefore, the pressure to reduce design cycle time and cost is enormous. One way to solve such a dilemma is to develop and apply advanced engineering environments (AEEs), which are distributed collaborative virtual design environments linking researchers, technologists, designers, etc., together by incorporating application tools and advanced computational, communications, and networking facilities.

Aircraft conceptual design, as the first design stage, provides major opportunity to compress design cycle time and is the cheapest place for making design changes. However, traditional aircraft conceptual design programs, which are monolithic programs, cannot provide satisfactory functionality to meet new design requirements due to the lack of domain flexibility and analysis scalability. Therefore, we are in need of the next generation aircraft conceptual design environment (NextADE). To build the NextADE, the framework and the data management problem are two major problems that need to be addressed at the forefront. Solving these two problems, particularly the data management problem, is the focus of this research.

In this dissertation, in light of AEEs, a distributed object-oriented framework is firstly formulated and tested for the NextADE. In order to improve interoperability and

simplify the integration of heterogeneous application tools, data management is one of the major problems that need to be tackled. To solve this problem, taking into account the characteristics of aircraft conceptual design data, a robust, extensible object-oriented data model is then proposed according to the distributed object-oriented framework. By overcoming the shortcomings of the traditional approach of modeling aircraft conceptual design data, this data model makes it possible to capture specific detailed information of aircraft conceptual design without sacrificing generality, which is one of the most desired features of a data model for aircraft conceptual design. Based upon this data model, a prototype of the data management system, which is one of the fundamental building blocks of the NextADE, is implemented utilizing the state of the art information technologies.

Using a general-purpose integration software package to demonstrate the efficacy of the proposed framework and the data management system, the NextADE is initially implemented by integrating the prototype of the data management system with other building blocks of the design environment, such as disciplinary analyses programs and mission analyses programs. As experiments, two case studies are conducted in the integrated design environments. One is based upon a simplified conceptual design of a notional conventional aircraft; the other is a simplified conceptual design of an unconventional aircraft. As a result of the experiments, the proposed framework and the data management approach are shown to be feasible solutions to the research problems.

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Motivation

In the competitive global market place, aerospace companies, like the ones in other high-tech industries, are forced to deliver the right products to the right market, with the right cost, and at the right time; while in the mean time, the need for the infusion and development of new technologies has dramatically increased the cost and complexity of designing an aircraft. Furthermore, through acquisitions and mergers most of these companies are now geographically distributed all over the world making collaboration extremely challenging. To address this challenge, current advances in the field of information technologies may be employed to develop a distributed collaborative virtual design environment. This environment will allow for the linking of researchers, technologists, designers, customers, etc., together by incorporating advanced computational, communications, and networking facilities and application tools.

Aircraft conceptual design, as the first stage of designing a new aircraft, plays a very important role in reducing design cycle time and costs. In this design stage, the concept of the new aircraft is formed through brainstorming various alternatives based upon historical data; the high-level structure of the whole complex system is created; targets on performance, cost, quality, safety, reliability, etc. are set. It is in the conceptual

design stage that the basic structure of product information is formed. The entire set of product information will be continually updated based upon this structure throughout the lifecycle of the aircraft. A well-known fact is that more than half of the production cost is committed during the conceptual design stage. It is believed that it is during this early design stage that one has the greatest opportunity for leverage and change. This is perhaps the time and the place where design changes can be made rapidly with little cost. The right decisions in this stage ensure a successful end product. This notion has gain general acceptance over the past decade and may be depicted graphically by the illustration in Figure 1 from Ref. [1] commonly referred to as the paradigm shift in design. In this figure, knowledge is information about the product and the process embedded in the context in which it was created; freedom is a measure of flexibility, the degree to which changes in product/process characteristic are viable; cost committed refers to the resource allocation determined through decision making. According to this figure, in the future, we will want to bring more design knowledge to the early design stages while keeping design freedom open longer and shifting to a more gradual cost committed curve**.** Thus, improving the efficiency (time) and effectiveness (cost) of aircraft conceptual design activities are critical.

However, aircraft conceptual design programs, which are currently used in practice, are unable to handle unconventional configurations or account for new technologies. This is due to their reliance on historical data, and perhaps in many cases, the inflexibility of the codes used to account for or to be modified to account for concepts and technologies outside the historical database, to create the relationships used throughout these programs. To account for these emerging needs, a new approach is

proposed which will yield a next generation aircraft conceptual design software environment (NextADE). It will enable us to shorten design cycle time, improve design quality, and reduce cost by allocating more design iterations or by incorporating higher fidelity tools at the conceptual design stage.

The design and development of such a design environment is an expensive and time-consuming process. There are still many research problems that need to be resolved. The research in this dissertation is an initial step to tackle the system's architectural and data management problems of the NextADE from the perspective of aerospace engineering by utilizing state of the art computing technologies.

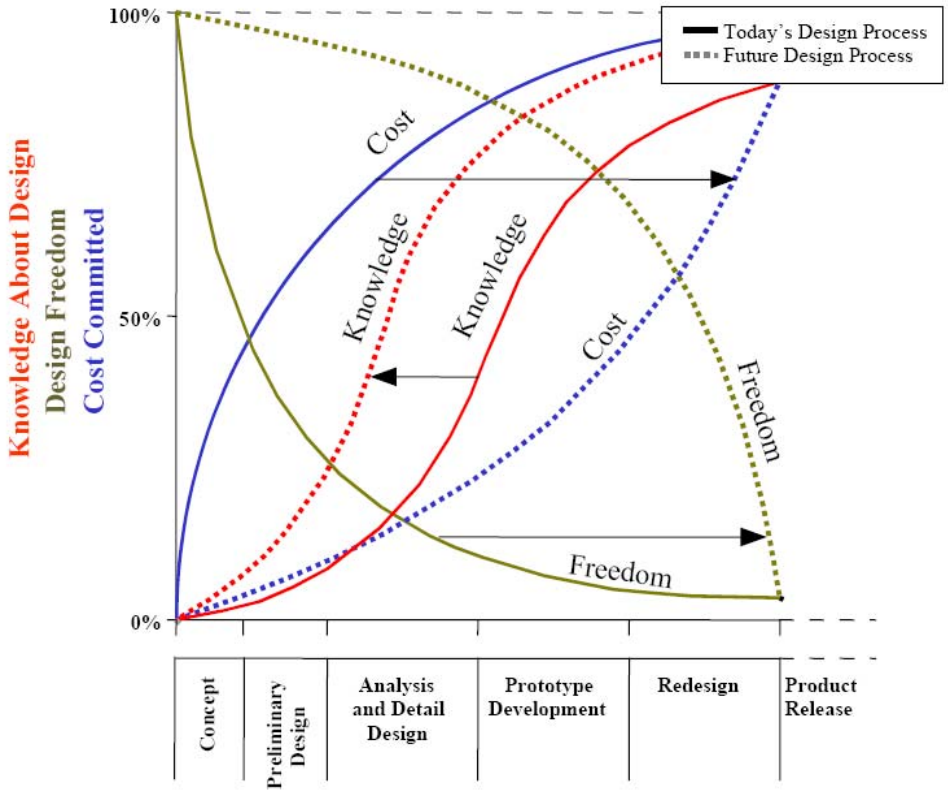

Figure 1. Paradigm Shift of Design [1]

Aircraft conceptual design is a computation-intensive, multidisciplinary, highly coupled, iterative decision-making process that requires the support of sophisticated computer-based design environments. The advancements in related disciplines, such as systems engineering, optimization, aerodynamics, propulsion, structures, controls, economics, etc., have dramatically increased the complexity of aircraft conceptual design at both of the system level and the component/subsystem level. The rapid developments of computing technologies over the last decades have also changed the way of how aircraft conceptual design is conducted. Many design activities heavily rely on virtual computer-based engineering simulations. Some companies have achieved significant accomplishments in utilizing these technologies; for example, the paperless design of the Boeing 777, which is the first airplane to be 100 percent designed using computer aided design (CAD) technologies. Today, like other design activities, aircraft conceptual design is being conducted in an extended enterprise environment. It is more of a collaborative endeavor that involves many individuals from different companies or different parts of a company, utilizing disciplinary analyses programs, which are distributed around the world. This fact adds additional challenges to the development of the NextADE. The NextADE should be a design environment that enables designers to streamline the use of multiple computing assets around the world.

In recent years, on both the academic side and the commercial side, there has been an explosion in the research and development of integrated virtual simulation environments, which aim to seamlessly integrate the dynamically interactive disciplinary analysis tools used in a design process. It has been widely recognized that the so-called "islands of automation" idea is a major problem that needs to be solved when integrating

different application software tools. This is a problem of data management regarding the integration and sharing of information among application tools and organizations.

An integrated virtual simulation environment usually consists of several to many application tools of different purposes. These heterogeneous application tools typically have been developed independently. They were written in different programming languages, work on different computer platforms, and are installed on different computers at different locations. Moreover, each of them has its own problem definition language and means for representing information. In a nutshell, these application tools are not interoperable. IEEE defined interoperability as the ability of two or more systems or components to exchange information and to use the information that has been exchanged [2]. The lack of interoperability among these application tools creates artificial barriers for communication and information sharing within the virtual integrated simulation environment.

There is no exception when it comes to aircraft conceptual design due to the fact that it is a complicated, data intensive, multidisciplinary, highly coupled, iterative design process. Most aircraft conceptual design programs that we are using in practice are treated in most cases as black boxes, which are based upon historical data. Flexibility is not provided for us to add new functionality and try new ideas that are beyond the coverage of historical data. Especially, design information management as the "glue" of linking different building blocks of these programs is not transparent. It is difficult and very costly to rewrite the codes especially if they are not well documented. In recent years, designers have started to use integration software programs to integrate contributing disciplinary application tools, making the design and analysis process more

and more automated. Using these software programs, design cycle time can and has been reduced, while design quality has been improved due to the automatic data passing and the so called “do it once and use it many times” approach to the problem. However, these improvements are limited. Actually with the exception of benefits provided through the automation of analyses processes, these software programs do not offer designers the functionality to effectively manage design information. The reason is that the integration is done based upon file parsing – a mere replacement of manual data passing from one application tool to another. From the viewpoint of information management, file parsing between different application tools is not a very effective means to manage design information of a complex system. Due to the heterogeneous nature of these disciplinary application programs, huge amounts of design information are sparsely spread in different input and output files of various forms, sizes, and computer platforms. The content of these files often overlaps. The structure and interpretation of them are also different. Therefore, recovering and reusing valuable design information contained in these files in most cases remains a daunting task. Obviously, as we formulate and develop the NextADE, emphasis will be placed in eliminating the communication barriers of integration as well as enabling design information reuse since both of these issues are very important and need to be addressed from the outset.

There is another fact that makes data management very important for aircraft conceptual design - past experience or historical data plays an essential role in the initialization of a new aircraft concept. It is often the case that a successful aircraft design is a direct development of earlier designs, i.e., an extrapolation of its predecessors. This means that the risk associated with this kind of design is much less if it is based upon a

large information repository of previous knowledge. Research efforts are being conducted to encapsulate past design knowledge in the form of knowledge-based computing [3]. In the long term, it is theoretically possible that artificial intelligence programs can accomplish the aircraft conceptual design process [4, 5]. However, in reality, this approach will not be mature enough to be utilized for quite sometime. As time goes by, the design experience and lessons learned will be condensed into "historical" databases. To build up these databases which will eventually become the future knowledge base, we need to firstly understand the structure of the information of aircraft conceptual design, and find effective ways to manage and use/reuse them in a computer-based design environment.

In summary, in order to improve the efficiency and effectiveness of aircraft conceptual design activities, improve the quality of design decision making, and reuse design information assets to a greater extent, it is time to take a step forward to develop the NextADE by incorporating the state of the art multidisciplinary system design methodologies and computing techniques. It is therefore the motivation behind the research in this dissertation to tackle the major issues associated with the architecture and the data management problems anticipated to be encountered during the design and development of the NextADE.

1.2 Thesis Organization

This thesis is organized into six chapters, each with a different focus. The first four chapters provide the introduction and background knowledge of aircraft conceptual design and enabling computing technologies. Research questions and hypotheses are posed in Chapter 2. Experimentations and their proposed solutions are detailed as such: the formulation of the proposed object-oriented distributed aircraft conceptual design environment framework and the proposed data management approach in such an environment (Chapter 5); the implementation results (Chapter 6); and experimentations with the proposed solutions (Chapter 7); finally, the last chapter, Chapter 8, summaries the concluding remarks, revisits the research questions and the hypotheses, and transitions to a set of recommendations for future work. These chapters are previewed next.

- *Chapter 1. Introduction and Background*

  Beginning with the motivation of this research, introduction and background information are provided in this chapter. The overall thesis organization is then introduced. A summary of the literature review performed to substantiate this research is also given.

- *Chapter 2. Research Questions and Hypotheses*

  The statement of research questions and hypotheses are made.

- *Chapter 3. Aircraft Conceptual Design*

  An introduction to traditional aircraft conceptual design is provided in this chapter. Literature on its latest developments over the past decade, such as the

probabilistic design methodology, is briefly reviewed. Principles of constraint analysis and mission analysis are also presented.

- *Chapter 4. A Brief Overview of Enabling Computing Technologies*

  The latest developments in computing technologies, especially information management technologies, are reviewed in search of potential enablers, which will help to solve the research problems. This chapter serves as a brief overview of related computing technologies, such as object-oriented design and analysis, and database management technologies.

- *Chapter 5. Formulation*

  The answers to the research questions are formulated. The proposed object-oriented distributed framework of the next generation aircraft conceptual design is presented first. Then, the proposed object-oriented data model, which lays the foundation for an effective data management approach for the NextADE, is described and explained in detail.

- *Chapter 6. Implementation*

  The implementation of the proposed solutions to the research questions is presented. The object-oriented mission analysis component is built up from scratch. A prototype data management system is developed for aircraft conceptual design based upon the proposed object-oriented data model.

- *Chapter 7. Experiments*

  Using a general-purpose integration software package, the integrated design environment is initially implemented in this chapter. Experiments based upon simplified conceptual design problems of a notional conventional aircraft and an

unconventional aircraft are conducted in this integrated design environment. Some observations made are presented at the end of the chapter.

- *Chapter 8. Closing Remarks and Recommendations*

  As the closure of this dissertation, the research is summarized. The research questions and the hypotheses are reinstated and answered. Recommendations for developing the approach further are presented.

1.3 Background

In order to facilitate the transition to the definitions of the research problems and hypotheses, the results of the literature review performed on the topic are presented first. This review serves as the foundation of the research reported in this dissertation. Section 1.3.1 will provide a brief overview of the state of the art in multidisciplinary design and analysis. In section 1.3.2, definitions of what are referred to as data and data management will be given. Critical issues for engineering data management will also be presented and discussed. As the guidelines of this research, some important research results on advanced engineering environments (AEEs) will be presented in section 1.3.4. Then, in section 1.3.5, the state-of-the-practice aircraft conceptual design programs will be investigated extensively so that we can see clearly what our needs are and what obstacles we need to overcome. Literature review on research works in related areas including integrated design environments and engineering data management will also be presented in section 1.3.6. Lastly, in section 1.3.7, commercially available integration software packages will be introduced. They are the commercial realization of research

achievements and a driving force in the advancement of research activities conducted in the academic world. As a part of the background knowledge body, introductions to traditional aircraft conceptual design and enabling state of the art computing technologies, particularly data management, will be provided with more details in Chapter 3 and Chapter 4 after the statement of research questions and hypotheses made in Chapter 2.

1.3.1 Multidisciplinary Design

Multidisciplinary Design Analysis/Optimization (MDA/O) was defined by Sobieszczanski-Sobieski as "*a methodology for the design of systems where interaction between several disciplines must be considered, and where the designer is free to significantly affect the system performance in more than one discipline*" [6]. Compared to single disciplinary optimization, the inherent interdisciplinary coupling in MDA/O increases complexity and creates additional challenges for implementing the necessary coupling in software systems. The implementation of these MDA/O procedures is limited by computational burden and the difficulty to integrate diverse models and associated analysis tools coming from different organizations.

The conceptual components of MDA/O include mathematical modeling of a system, design oriented analysis, approximation concepts, system sensitivity analysis, optimization procedures with approximation and decomposition, and human interface [7]. Among these elements, the approximation and decomposition usually determine the procedure organization. Decomposition refers to partitioning a large task of the engineering system synthesis into smaller tasks without being limited to formalism of the

top-down, hierarchical decomposition. The advantages of decomposition lay in that the smaller tasks tend to be aligned with existing engineering specialties, and it also provides the opportunities for concurrent operations [6, 8].

1.3.2 Object-Oriented Design and Analysis Philosophy

Object-oriented concepts originated in computer science in the early 1970's. Today, object-oriented techniques are not only being used widely in software engineering but are increasingly being exploited as design and analysis methods because of their abilities to easily cope with the complexity inherent in many different kinds of systems.

As defined by Booch, object-oriented design is *"a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design*" [9]. Object-oriented analysis is "*a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain*" [9].

Unlike the procedural approach, which is algorithm oriented and asks the question of what this program does, the object-oriented approach using class and object as basic building blocks is based upon the question of what real-world objects one is modeling. It offers a variety of attractive features, such as encapsulation, modularity, inheritance, polymorphism, etc. As stated by Booch, "*object-oriented analysis and design may be the only method that we have today that can be employed to attack the complexity inherent in very complex system*s" [9].

1.3.3 Data and Data Management

In general, unlike a physical entity, information is an abstract concept which is used in a context and in relation to some reasons. Webster defines information science as "*the collection, classification, storage, retrieval, and dissemination of recorded knowledge treated both as a pure and as an applied science*" [10]. More specifically, with the emphasis on the need for and use of information, Saracevic defines information science as "*a field of professional practice and scientific inquiry addressing the problem of effective communication of knowledge records – 'literature'- among humans in the context of social, organizational, and individual need for and use of information*" [11].

Within the field of information science, data is seen as something raw, on a path to being fully cooked or distilled, or in a sequence from data to information to knowledge to wisdom [12]. In this dissertation, data refers to information gathered for processing or decision-making [13]. Therefore, the distinction between data and information is not very strict. They are interchangeable in this dissertation.

Data management is the ability to organize information, usually through database management systems. It encompasses functions of controlling the acquisition, analysis, storage, retrieval, and distribution of data. It involves:

- Establishing a data architecture, including creating information models
- Reducing redundancy in data
- Protecting confidential or private information in data
- Protecting the physical security of data, and
- Ensuring back up and recovery procedures are in place

There have been three generations of data management systems: data files, relational databases, and object-oriented databases. Each of them has advantages and disadvantages. Brief introductions to these systems will be provided in Chapter 4.

1.3.4 Advanced Engineering Environments

The concept of the advanced engineering environment (AEE) emerged circa 1999. It has been recognized that, based upon the successful experience in the application of computer aided design, engineering, and manufacturing systems in the last decade, we now "*have a historic opportunity to develop and deploy AEE technologies and systems*" [14, 15]. A two-phase study of AEEs, which was sponsored by the National Aeronautics and Space Administration (NASA), has been conducted by a committee appointed by the National Academy of Science, the National Academy of Engineering, and the Institute of Medicine. The Phase 1 study focuses on the short-term in identifying and assessing the needs, directions, and barriers during the next 5 years for the development and implementation of AEEs in a national framework. The Phase 2 expands the assessment to the long-term (5 to 15 years) vision for incorporating AEE technologies and systems into both the current and future engineering work force.

According to the Phase 1 report of the research, AEEs (i.e., AEE systems) are "*particular implementations of computational and communications systems that create integrated virtual and/or distributed environments linking researchers, technologists, designers, manufacturers, suppliers, and customers involved in mission-oriented, leading-edge engineering teams in industry, government, and academia*" [14]. An AEE

includes three key components: computation, modeling, and software; human-centered computing; and hardware and networks. The committee on AEEs envisioned that an ideal AEE would possess the following capabilities [14]:

- Encompassing concept definition, design, manufacturing, production, and analysis of reliability and cost over the entire life cycle of a product or mission in a seamless blend of disciplinary functions and activities
- Making it easier to implement innovative concepts and solutions while effortlessly drawing on legacy data, tools, and capabilities. Interoperability would not require burdensome development of new software to provide customized interfaces
- Accommodating a diverse user group and facilitating their collaboration in a manner that would obviate cultural barriers among different organizations, disciplines, and geographic regions
- Including a high speed communication network for rapid evaluations of concepts and approaches across engineering, manufacturing, productions, reliability, and cost parameters with high fidelity

Based on surveys conducted in industry and government organizations, the committee identified a number of technical, management, cultural, and educational barriers that need to be overcome first in order to realize the vision of AEEs. On the technical side, the common problems observed related to the integration of tools, systems, and data, and information management. A detail list of those recognized technical barriers is given in Table 1.

Table 1. Technical Barriers to Achieving the AEE Vision (Adapted from [14])

Integration of Tools, Systems, and Data	
1	Lack of tool interoperability
2	Continued proliferation of tools, which aggravates interoperability issues
3	Existing investments in legacy systems and the difficulty of integrating legacy systems with advanced tools that support AEE capabilities
4	Little effort by most software vendors to address interoperability or data-exchange issues outside of their own suite of tools
5	Multiple hardware platform issues - computers, hardware, databases, and operating systems
6	Lack of formal or informal standards for interfaces, files, and data terminology
7	Increasing complexity of the tools that would support AEE capabilities
8	Difficulty to inserting emerging and advanced technologies, tools, and processes into current product and service environments
9	Supplier integration issues
10	Difficulty of integrating AEE technologies and systems with other industry-wide initiatives, such as product data management, enterprise resource management, design for manufacturability/assembly, and supply-chain management
Information Management	
1	Proliferation of all types of information, which makes it difficult to identify and separate important information from the flood of available information
2	Difficulty of maintaining configuration management for product designs, processes, and resources
3	Need to provide system "agility" so that different types of users can easily input, extract, understand, move, change, and store data using familiar formats and terminology
4	Difficulty of upgrading internal infrastructures to support large bandwidths associated with sharing of data and information
5	Need to provide system security and to protect proprietary data without degrading system efficiency

Depending on granularity, people may decompose the life cycle of a product and mission slightly differently. In the Phase 2 report, the AEE committee divided the entire life cycle of a product and mission up to eight stages. They are listed in a chronologically order [15]:

1) Mission requirements analysis/product system strategy
2) Product specification
3) Concept development
4) Preliminary product and process design
5) Refinement and verification of detailed product and process designs
6) System prototype development
7) Production, testing, certification, and delivery
8) Operation, support, decommissioning, and disposal

The concept development stage is all about target setting for cost, performance, schedule, etc., brainstorming on product and process alternatives, and the development of product and process concepts. This is in line with how we previously defined aircraft conceptual design.

Assuming that the future of computer applications and technology is foreseeable on a 15-year time scale, the AEE committee addressed the development of AEEs over the next 15 years at each stage of the entire life cycle of products and missions. At the concept development stage, today's typical practice of evaluating alternatives tends to be driven by first-order analytical tools, rules of thumb, and existing data based on benchmarks. The AEE committee envisioned that these practices would be changed dramatically in the future. The long term goals set up by the committee for conceptual

development are summarized in Table 2 [15]. It has also been suggested that for today's practice it is possible to use available performance data more systematically.

Some AEE technologies are available and being deployed, and some will be discovered and developed. In the short term, we might not be able to realize the AEE vision. However, in the long run, when we design and develop new complex design systems, we should keep them consistent with the vision of AEEs. The ideal capabilities of AEEs described in this section are used as the guidelines of the research efforts in design and developing the NextADE reported in this dissertation.

Table 2. Future Practices of the Concept Development Stage [15]

2015 Vision: Integrated Manufacturing Technology Roadmapping Initiative
- Integrated, predictive life-cycle cost and profitability models
- Optimization of shared resources
15-Year Vision: NASA's Intelligent Synthesis Environment Initiative
- Complete life-cycle optimizations, trading performance, cost, risk, and schedule
Future Perfect: Conceptual design, preliminary design, and detail design combined
- Expert systems generate alternatives
- Optimized, top-down concept development process
- Automatic analytical evaluation of all product and process attributes (including risk and uncertainty)
- Single-pass product and process design and concurrent evaluation with multifunction optimization and automatic cascade to next lower level of design
- Automated generation of details about component and subsystem design and manufacturing details from high-level descriptions and desired attributes
- Single data source
- Full automation for subsystem and component tracking and trade-offs
- No late trade-offs

1.3.5 Legacy Aircraft Conceptual Design Programs

In this section, the state of the practice in aircraft conceptual design is provided. It includes an introduction of legacy aircraft conceptual design programs that we are using in practice, and their limitations in meeting current and emerging requirements. At the end of this section, the desired traits of the NextADE are proposed. Please see Chapter 3 for the basic knowledge of aircraft conceptual design, e.g. the design process, and the design principles, etc.

There is a plethora of aircraft conceptual design programs in use in the public domain and/or government owned for a range of aerospace vehicle type applications. All companies have similar capabilities based on propriety information. Since they are not available publicly, they are beyond the scope of the review and investigation. Following is a partial list of the current state-of-the-practice aircraft conceptual design programs found in the open literature and which may be classified as public-domain and/or government owned:

- For commercial/military fixed-wing aircraft:
    - FLOPS: Flight Optimization System
    - ACSYNT: Aircraft Synthesis Program
    - IDAS: Integrated Design & Analysis System
- For tilt rotor/VSTOL aircrafts:
    - VASCOMP: V/STOL Aircraft Sizing & Performance Computer Program
    - ACSYNT: Aircraft Synthesis Program

- For rotorcraft:
    - GTPDP: Georgia Tech Preliminary Design Program
    - HESCOMP: Helicopter Sizing & Performance Computer Program
- For general aviation:
    - GASP: General Aviation Synthesis Program
- For space access/hypersonic vehicles:
    - HAVOC: Hypersonic Aircraft Vehicle Optimization Code

Some of these programs date back to the 1960s. They may be characterized as monolithic codes, written in a procedural programming language, for example, FORTRAN. These programs have been proven to be highly effective for the purposes of designing aircraft within their prescribed domains (i.e., interpolation within the historical database they are based on). Therefore, they are highly trusted. Furthermore, some of them have been under steady improvement over the past decade, notably FLOPS and ACSYNT, which represent the first generation integrated aircraft synthesis tools.

FLOPS was developed by NASA Langley Research Center. It is a multidisciplinary sizing and synthesis code for conceptual and preliminary design and the performance evaluation of conventional aircraft concepts [16]. Like many other legacy design programs, FLOPS was written in FORTRAN. It consists of nine primary modules: weights, aerodynamics, engine cycle analysis, propulsion data scaling and interpolation, mission performance, takeoff and landing, noise footprints, cost analysis, and program control. Through the program control module, FLOPS maybe used to analyze a point

design, parametrically vary certain design variables, or optimize a configuration with respect to these design variables using nonlinear programming techniques. For optimization, the Kreisselmier-Steinhauser (KS) [17] function or the Fiacco-McCormick [18] penalty function may be used with the Broyden-Fletcher-Goldfarb-Shano (BFGS) algorithm.

ACSYNT is another aircraft conceptual design environment that is representative of the state-of-the-practice. ACSYNT is an abbreviation for AirCraft SYNThesis. This program was originally developed during the early 1970s by the NASA Ames Research Center, which focuses on conceptual design studies of advanced aircraft [19]. A highlight of ACSYNT is that it is one of the first aircraft synthesis codes designed in a modular fashion. This program architecture has allowed more flexibility compared to other design programs. ACSYNT comprises several discipline codes that can either be standalone for analysis of one aspect or be combined with other modules in order to evaluate the integrated results. The disciplines represented include modules for geometry, weights and structures, aerodynamics, propulsion, mission performance, cost, takeoff performance, and sonic boom. Actually, decomposed disciplinary modules, such as geometry, aerodynamic and propulsion, are used in the implementation example of this research work in later chapters. Most of the analyses are written in FORTRAN and some are in C and C++. Much of the geometry modeling is in C++.

The structure of ACSYNT is depicted in Figure 2. The program has a control module that determines the execution sequence coupled with an optimization module. The analysis modules are connected to the control module through a data management routine. The geometry module of ASCYNT is used to model the geometry of the aircraft,

produce a displayable image and provide information (e.g., surface area and volume) to perform aerodynamic calculations (e.g., lift and wave drag calculations). The trajectory module evaluates the mission and point performance of a design by interacting with other modules. The fidelity of each disciplinary analysis is on the same level as those of FLOPS, i.e., conceptual design level. As mentioned previously that each of these programs works well in its specific domain, FLOPS is usually considered to be better for modeling transport vehicles and ACSYNT is better for fighter configurations. A non-linear optimization code, the COPES/CONMIN [20] optimization package, can be included in ACSYNT so that it can optimize a vehicle design for a particular objective function, e.g. minimize the takeoff gross weight, subject to various constraints. An interactive computer aided design (CAD) interface to ACSYNT was also created and this enables the execution and control of the design process via interactive graphic menus and allows

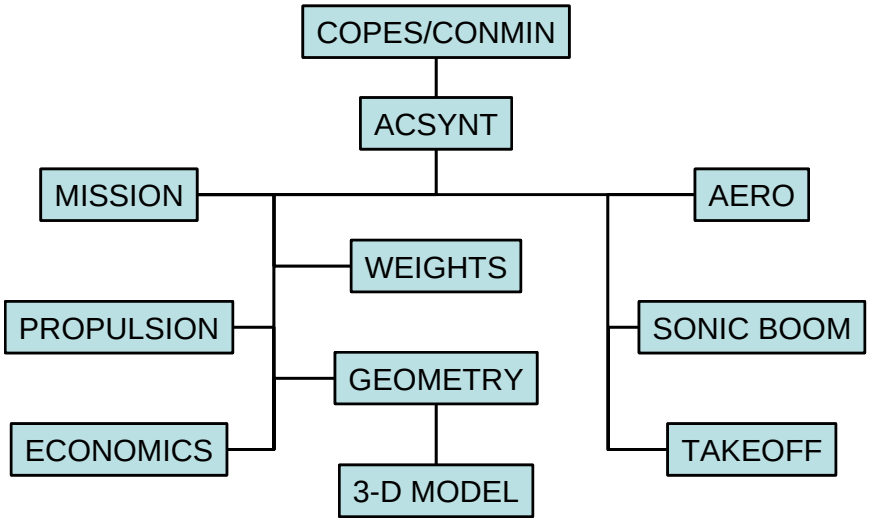

Figure 2. ACSYNT Program Module Structure [19]

rapid evaluation of design configurations. However, this CAD system was coded with the 3-D graphics standard, Programmer's Hierarchical Interactive Graphics System (PHIGS), which limits the CAD version of ACSYNT to work only on workstations.

These legacy aircraft conceptual design programs were at the cutting edge when they were developed. However, these tools in their present form are inadequate in meeting the current requirements imposed by the need to design and analyze unconventional vehicles which lie outside the historical databases that these codes are based upon, i.e., the design and analyses of such vehicles are an extrapolation of the historical data. The major challenge we are facing is that due to the lack of flexibility, scalability and openness of system architectures, they are very difficult to reuse or rewrite to accommodate current needs, although some of them are well documented. This is actually a double-edged sword. On one side, it makes these programs efficient and trustable within their specific application domain. On the other side, it makes it difficult to incorporate innovative methods and new technologies, particularly for the conceptual design of unconventional concept vehicles.

Unconventional concept vehicles refer to future air vehicles that look, behave, and operate fundamentally differently than those in current and past experience [21]. For example, air vehicles with unconventional geometries and performance capabilities, air vehicles powered by hybrid and distributed propulsion systems, and fixed wing vehicles that achieve vertical/short takeoff and landing capabilities, etc. As such vehicles become prominent in forecasting future military and commercial applications, weaknesses in the state-of-the-practice design programs for their design are becoming apparent. Two of the

major limitations exhibited by these legacy aircraft conceptual design programs will be discussed next.

First, concerning domain flexibility, the state-of-the-practice aircraft sizing programs are usually only suitable for a few specific aircraft types because the design algorithm is hard coded into the program. These design algorithms represent physics that are restrictive to particular types of vehicles and do not allow for customization to accommodate new algorithms required for unconventional concept vehicles, e.g. the sizing algorithm for air vehicles using non-consumable fuels, i.e. fuel cells. Further, these programs do not provide the functionality to model a wide variety of mission profiles. A view into the domain scalability challenge is presented in Table 3 for the propulsion perspective.

Table 3. Domain Scalability - Propulsion Example

New Domain	Challenge to sizing algorithm
Alternative dissipative fuels (e.g., H_2)	Modifying "fuel balance" routine
Non- Dissipative fuels (e.g., solar)	Energy-balance routine; modified wing sizing algorithm
Hybrid configurations (e.g., solar/fuel)	Energy and fuel balance

Second, concerning analysis scalability, the state-of-the-practice aircraft conceptual design programs are usually monolithic codes and non-scalable. The contributing disciplinary analyses programs, such as aerodynamics, engine cycle analyses, and cost analyses, are fully integrated inside the conceptual design programs. Without tedious, time-consuming restructuring, rewriting, and recompiling, it is impossible to add or update an analysis function to the design programs. This makes the design programs

themselves “black boxes”. Designers do not have the flexibility to use different disciplinary analysis programs other than those built-in ones. Usually the built-in disciplinary analyses functions are at the conceptual level. Therefore, it is impossible for designers to perform variable fidelity disciplinary analyses when needed. Consequently, the latest and most up-to-date advances in the contributing disciplines cannot be fully utilized to support the conceptual design and analysis activities. A view of analysis scalability challenges is presented in Table 4.

Table 4. Analysis Scalability - Aerodynamics Example

Aero Analysis		Challenge to sizing process
Specified drag polar	↓	
Semi-empirical analysis	↓	Increasing level of fidelity brings integration difficulties for the contributing disciplinary analyses and possible mismatch errors in the data management scheme
Vortex lattice analysis	↓	
Euler analysis		

In addition to domain flexibility and analysis scalability, these design programs do not support distributed collaborative design. They are also platform dependent. As to data management, the use of common variables is the common way for data sharing among different analysis functions built within these design programs. Compared to other programs, it is valuable to see that ACSYNT goes a bit further on data management. ACSYNT manages data parsing among different modules using a center data repository together with common variables. It is an early attempt to solve the integration problem of an aircraft conceptual design computer program by utilizing a centralized data

management approach. However, the center data repository is just a flat file containing only numbers without any semantic context. Without knowing how the data file was originally designed and structured, it is very difficult for users to know which number corresponds to which variable or attribute, and vise versa. Also, functionality is not provided for managing this data file. The data file would be meaningless if used separately from the design program. Obviously, as the design itself and the design environments become more and more complex, to improve interoperability of the heterogeneous disciplinary application programs involved, a data management approach should offer more capabilities than what we have now. Effective and efficient data management is one of the fundamental and critical issues that need to be addressed in designing and developing the NextADE.

Based on the above discussion, we can see that the NextADE must include traits such as flexibility, scalability, interoperability, efficiency, robustness, extensibility, and reusability, as listed in Table 5. Current state-of-the-practice aircraft conceptual design programs have achieved advances in some of these categories. However, it is now apparent that this has been accomplished at the price of flexibility, scalability, and interoperability. These are the barriers that need to be overcome to meet the future design requirements in the conceptual design stage, for example, the design of unconventional concept vehicles and the requirement for higher fidelity disciplinary analyses.

Table 5. Desired Traits of the NextADE

Ideal Trait	Definition
Efficiency	The ability to do a task within certain constraints, such as time and memory space
Robustness	The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [2]
Scalability	The ease with which a system or component can be modified to fit the problem area
Flexibility	The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [2]
Extensibility	The degree to which an application or component is able to be enhanced in the future to meet changing requirements or goals
Interoperability	The ability of two or more systems or components to exchange information and to use the information that has been exchanged [2]
Reusability	The characteristics that codes written for one application can be reused in different applications

1.3.6 Related Research Work

In other areas, designers are experiencing the same kind of problems described in the previous section. In the last decade, a considerable amount of research works has been carried out in aerospace and related areas to address these problems. These research works laid a solid foundation for the author to understand and solve the research questions. In this section, some of the highlights of the literature review are provided to explore the depth and the breadth of these research activities and their respective achievements.

1.3.6.1 Integrated Design and Analysis Environments

In 1993, a new design formulation was presented by Stephens, who stated that a design can be represented by a centralized definition which encapsulates a distributed instantiation [22]. The concept of "a centralized definition" is a very important concept that inspires the author on solving the problem of how the data management system should be designed for the NextADE. According to Stephens, a distributed instantiation refers to a design that is evaluated using supportive resources that are distributed organizationally and geographically. In his research work, focusing on design information, process strategy, and design evaluation, Stephens recognized that a centralized definition for a design consists of four types of design components: functions, forms, models, and processes. All these design components are described parametrically and arranged in multiply connected hierarchies through schemas, frames, and attributes.

It has been shown that hierarchical design components can be used to represent a design space and describe the design's capabilities, its components, how it works, and why it operates that way. Also design process strategies, which were represented by intellectual process models and implemented using agents, were developed as protocol descriptions with described tasks within the design space. To demonstrate this formulation, LEGEND, a Laboratory Environment for the Generation, Evaluation, and Navigation of Design, has been constructed from his work.

In 1995, the Air Force Research Laboratory identified the need to form an advanced conceptual aerospace assessment capability for future military aircraft systems. In 1996, the Air Force Research Laboratory, Aeronautical Systems Center, and Georgia Tech conducted a survey amongst government and industry users about the efficacy of state-of-practice tools and the desired requirements for future development efforts. As a result, a module architecture which enables variable fidelity analysis as shown in Figure 3 was proposed and a baseline analysis tool set was selected from those in use by industry and government [23]. This baseline tool set and its implementation is known as the Conceptual Aerospace System Design and Analysis Toolkit (CASDAT). The goal of CASDAT is to build a distributed design system in which the selected tools could be executed in a heterogeneous environment, necessitated because of the reliance on legacy tools that exist only on specific platforms. An internet-enabled software infrastructure called the Intelligent Multidisciplinary Aircraft Generation Environment (IMAGE), developed by Hale et al. in the Aerospace Systems Design Laboratory of Georgia Institute of Technology was used as the framework of CASDAT. The IMAGE infrastructure is depicted in Figure 4. It is built upon agent-based technologies and aims

to enable designers to make the best use of available assets to support the design activities using a computational backplane [24-26].

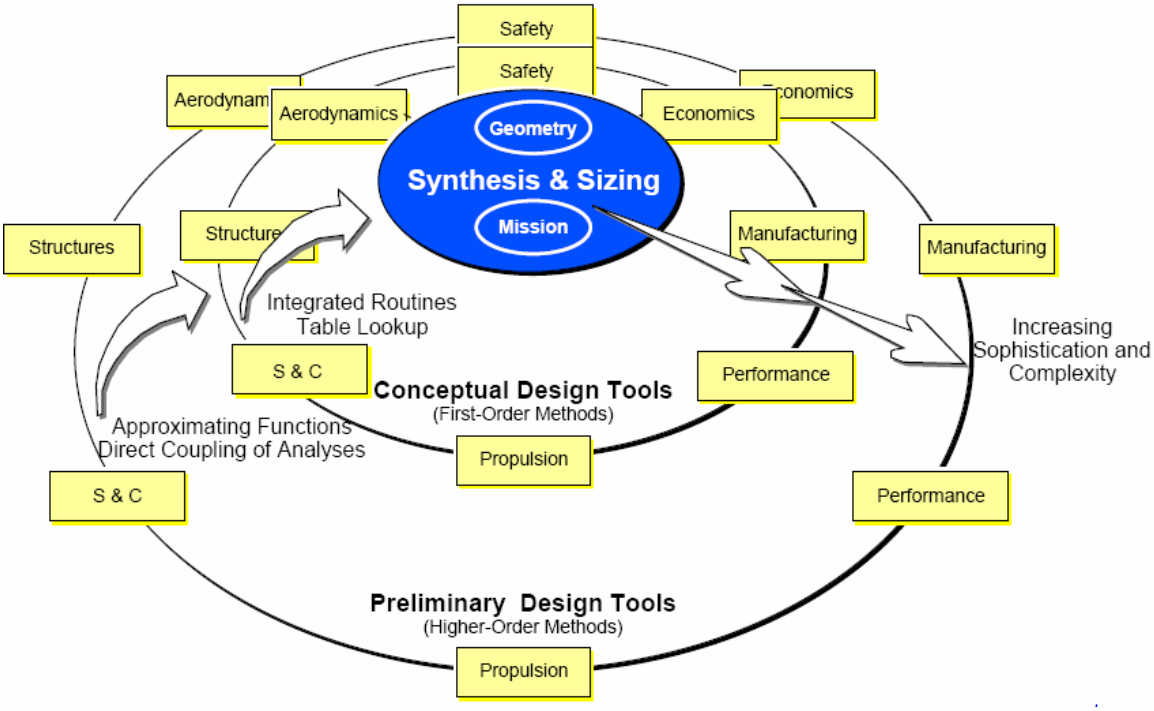

Figure 3. Synthesis and Sizing Architecture in CASDAT [23]

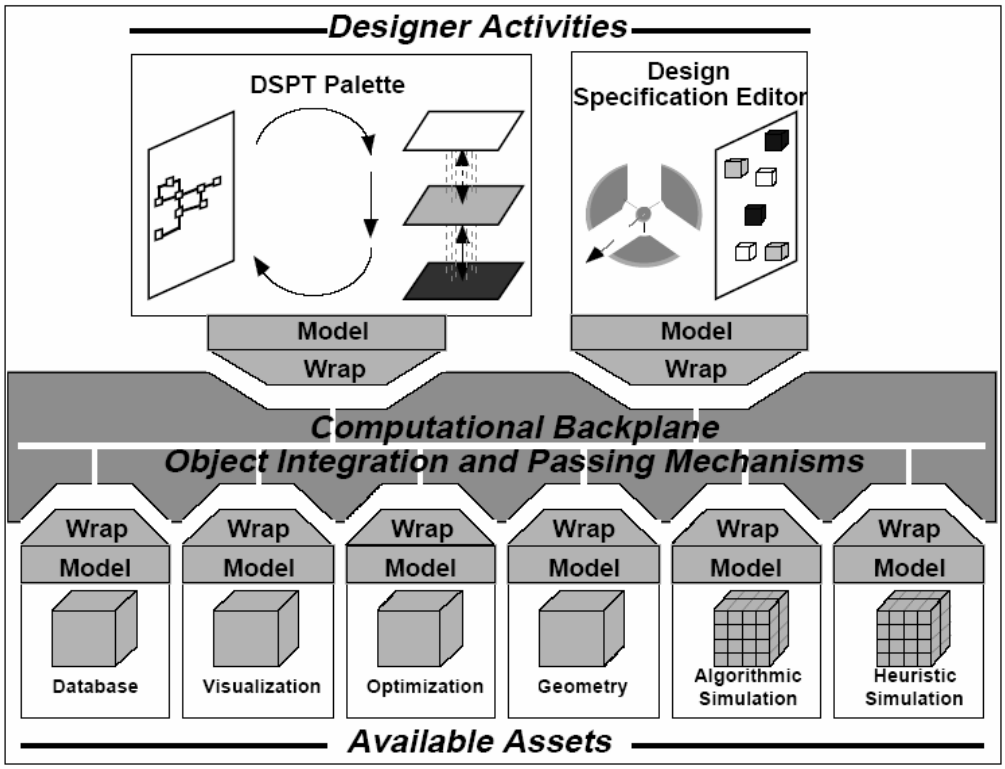

Figure 4. IMAGE Infrastructure [25]

The above infrastructure utilizes an object-oriented approach. In his 1990 thesis, Kolb conducted an investigation on the feasibility of applying object-oriented programming and constraint propagation to the development of a general-purpose computer aid for engineering conceptual design with an aim of improving the flexibility of such computer aided capabilities [27]. It had been realized that computer aids of engineering conceptual design demand extreme flexibility due to the facts of 1) conceptual design being unpredictable, and 2) conceptual design often seeking to take advantage of the latest advances in the state of the art. The basis of his approach is

component modeling, which organizes design knowledge according to the physical objects. Component descriptions are based on attributes and constraints. Attributes refer to the parameters describing the properties of a component; constraints are the mathematical relationships among the parameters. It has been demonstrated in the research that the use of object-oriented programming techniques, which support the definition of a new component through combination and modification of existent types, can simplify component definition. On the other hand, the adoption of constraint propagation, which accommodates the evolving problem description, is helpful to add flexibility to mathematical analyses. The author also advocated that design knowledge in the form of mathematical relationships governing a design should not be treated as codes to be executed but data to be manipulated by computers.

The rapid development of computing technologies over the last decade makes it possible to build sophisticated design and analysis environments for complex systems. An increasing amount of research work has been done or is being conducted in the design of complex systems; for example, ship design and propulsion system design. The research results and lessons learned are foundational and have influenced the formulation and development of the proposed NextADE.

The Leading Edge Advanced Prototyping for Ships (LEAPS) system, developed by the Carderock Division of Naval Surface Warfare Center (NSWC), is an example of an object-oriented environment created to integrate modeling, simulation and analysis tools [28]. Its primary objective is to enable rapid evaluation of design alternatives in an expanded trade space for product quality and interoperability. Based on a structured database that incorporates the interrelationships among different aspects of ship structural

characteristics, LEAPS facilitates integration of software across multiple disciplines, including mission requirements, ship design, engineering, costing, and warfare analysis. Information is shared among subject matter experts performing design and technology studies. The LEAPS environment, as shown in Figure 5, consists of a generic object-oriented class structure for data modeling, and an application programmer interface (API) for accessing the data model, translators between the new digital ship models and existing modeling, simulation and analysis tools used in naval design and acquisition, and other infrastructure tools such as viewers, editors, and input and output utilities. In the environment shown in the figure, ASSET is an abbreviation of Advanced Surface Ship Evaluation Tool, which is a family of ship design synthesis programs.

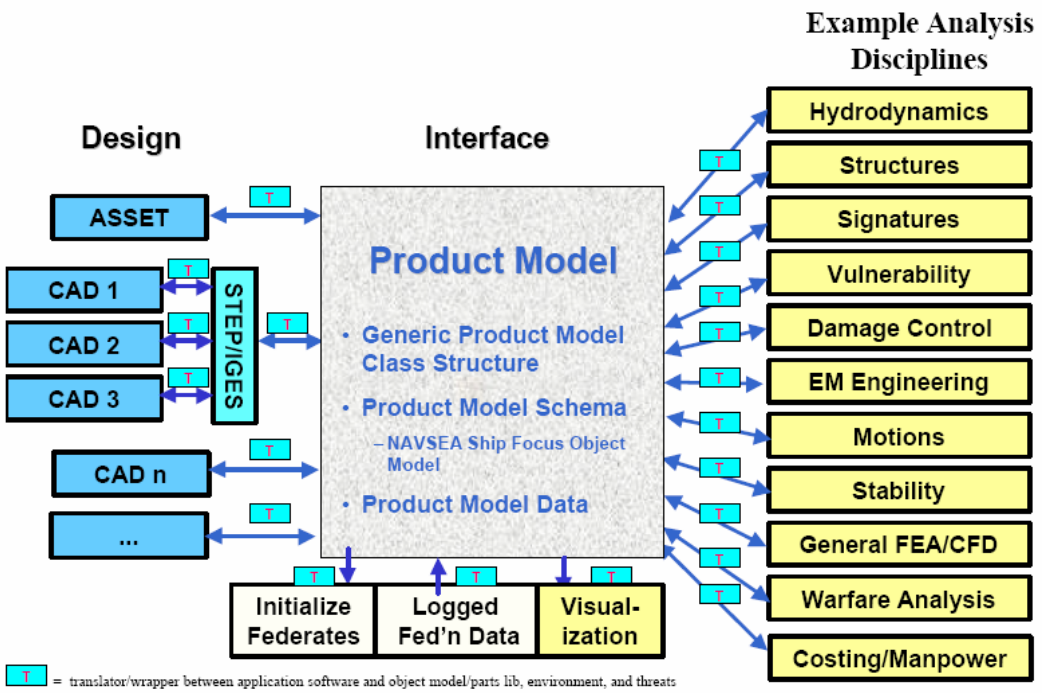

Figure 5. Leading Edge Advanced Prototyping System/Ships Environment [29]

Besides how different disciplinary analyses programs can be integrated into the design environment as objects, the architecture of LEAPS provides the author some insights on the effectiveness of data management in a sophisticated advanced design environment in order to improve interoperability among the heterogeneous contributing disciplinary analyses, and one of the potential ways to solve the data management problem in the NextADE. Overall, effective data management is essential in the NextADE.

Another design program that is being developed and can represent the state of the art is the Numerical Propulsion Simulation System (NPSS), although it seems that the data management issue in this simulation system is not addressed particularly. NPSS is a propulsion system simulation tool used to predict and analyze the aerothermodynamic behavior of commercial jet aircraft as well as military and space transportation [30, 31]. NASA's High Performance Computing and Communications Program supported the development of NPSS. NPSS is being developed through a NASA/Industry Cooperative Effort agreement between NASA Glenn and its industry and Government partners. The focus of NPSS is on the integration of multiple disciplines such as aerodynamics, structure, and heat transfer with numerical zooming on component codes. The program takes parameters such as Mach number, altitude, and mass flow along with component parameters and generates thermodynamic data for every point in the engine. An object-oriented approach was chosen for NPSS because it allows new codes to be introduced into the system easily. Shown in Figure 6 is an example object-oriented

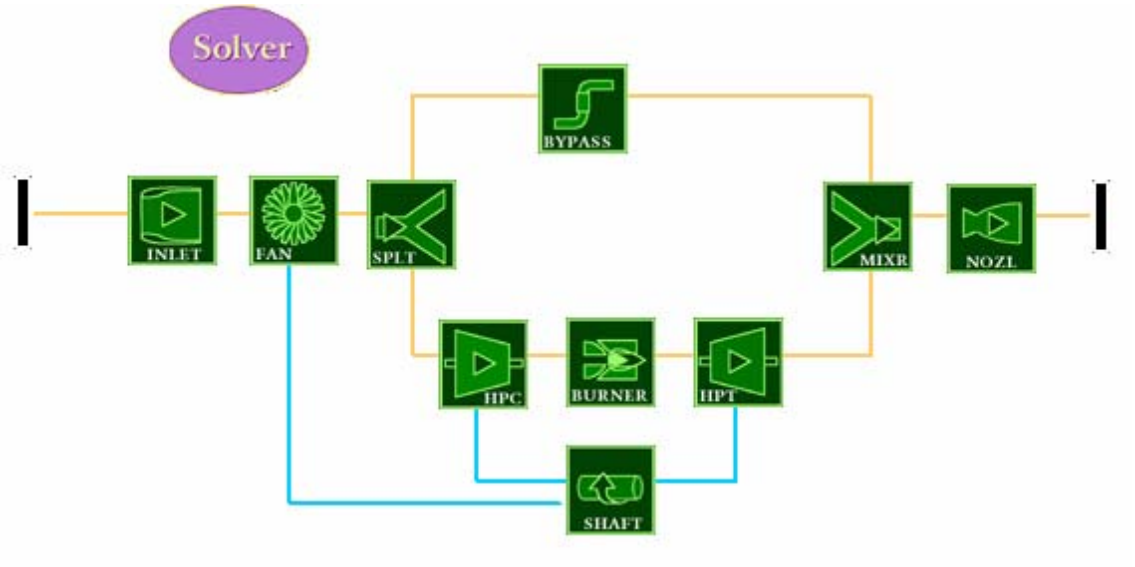

Figure 6. NPSS Object-Oriented Decomposition [30]

decomposition of turbo fan engines in the NPSS systems. The object-oriented approach lends itself naturally to the decomposition of the whole engine. The goal of NPSS is to create a numerical "test cell" that enables designers to create complete engine simulations overnight on cost-effective computing platforms. Using NPSS, as one of the benefits coming along with its object-oriented design, engine designers would be able to analyze different parts of the engine simultaneously, perform different types of analysis simultaneously (e.g., aerodynamic and structural), and perform analysis in a more efficient and less costly manner. Thus, the development time of a new engine can be cut in half, from 10 years to 5 years.

In 2001, NPSS won NASA's 2001 Software of the Year award. The success of NPSS gives the author confidence in adopting an object-oriented approach to design the NextADE due to the high level of similarity (besides the difference of domains) between

the multidisciplinary design and analyses of a propulsion system and the conceptual design of an aircraft. On a very high level, one distinctive feature that differentiates aircraft conceptual design from propulsion system design is that, in addition to component design, aircraft conceptual design is also based upon a mission profile. Unlike the physical components of a propulsion system or an aircraft, the mission profile of an aircraft is an abstract concept. However, we can still model the mission of an aircraft object-orientedly is similar to how NPSS models the components of a propulsion system. In Chapter 5, how the author solved the problem will be presented and explained in detail.

Before the AEE concepts, in 1994, NASA's Jet Propulsion Laboratory (JPL) assembled the advance Concept Design Team, know as Team X, in order to create a streamlined and fast-track approach to initial mission conceptualization and system architecture design [32]. The team consists of spacecraft and instrument designers and launch vehicle, trajectory, and orbit, ground system, mission planning and costing experts. Team X works concurrently with the proposers in developing scenarios and responds them with detailed integrated mission/instrument design and development plan. By adopting distributed spreadsheet models for spacecraft sizing, Team X effectively cut its cost to produce a proposal and reduced the turnaround time [33].

1.3.6.2 Engineering Data Management

Besides the development of these complex design and analysis environments, there is also a considerable amount of research activity that focus on engineering data management. Back in the 1980's, some researchers started to work on engineering data

management. It was first identified that there was not a suitable data modeling technology for engineering data [34, 35]. According to Morris et al [36], the following is a summary of some of the distinctive characteristics of engineering data:

- The structure of engineering data is non-uniform and unpredictable
- Many of the commonly used concepts require representations which are networks of data structures and relationships
- The interconnections between data structure are numerous and the same data structure may participate in many roles
- A large percentage of data is dependent on the existence of other data
- Completeness of a data set is relative to the stage in a product's life-cycle
- The level of accuracy needed for numeric values can vary depending on the semantics of the data and the application using the data
- Complex rules exist for data instantiation
- Algorithms may be required to ensure data integrity
- A single abstract concept may sometimes be represented at a detailed level in more than one way
- The evolution of a product or design is an important historical record

With the rapid development of technologies over the last decade, data-modeling technology itself is not a big problem any more today. Rather, data sharing among heterogeneous application programs is a research direction. Modeling the data in a specific domain according to its logic for reusing and effectively integrating the application programs in other domains is a challenge to be conquered. Such problem

originated from promoting concurrent engineering in the early 1990's [37]. Coming next are the highlights of some related research works.

In 1992, Peak et al proposed using Product Model-Based Analysis Models (PBAM) for data exchanged between disparate analysis methods [38, 39]. A PBAM is defined as "*a representation of engineering analysis models that includes linkages to product model design information*" [38, 39]. The representation includes both defined structure and defined operations. PBAMs can be created by filling the defined structure with analysis model-specific information. They are viewed as constraint schematics, which are constraints and constraint graphs to engineering analysis. PBAMs combine objects and constraints and enable routine analysis concurrent with product design. In another paper Peak et al. published in 1998, a multi-representation architecture (MRA) was proposed in order to bridge the gap between CAD and computer aided engineering (CAE) [39]. MRA was described as a design-analysis integration strategy that integrates CAD-CAE by an information-intensive mapping between design models and analysis model. Together with PBAMs, there are other three information representations used in the MRA: solution method models, analysis building blocks, and product models. Again, object and constraint graph techniques are used for modularity and data semantics.

In their 1997 paper, Hall and Fulton presented their research result on the impact of data modeling and database implementation methods on the optimization of conceptual aircraft design [40]. Focusing on the stability and control analysis portion of aircraft conceptual design, they investigated two database design approaches. One is the IDEF1X [41] approach with a relational implementation. IDEF stands for integrated

definition methods. IDEF1X is a method for designing relational databases with a syntax designed to support semantic constructs for developing a conceptual schema. The other approach is an EXPRESS approach with a C++ programming language implementation. EXPRESS [42] is another language for the definition of data schemas, which is a part of the Standard for the Exchange of Product Model Data (STEP) [43, 44]. The major conclusion drawn from this study is that conceptually, both the IDEF1X/relational and EXPRESS/object-oriented approaches fall short of providing users with effective schema change capabilities, i.e. data model changes. However, the EXPRESS/object-oriented approach requires less work, which is a desirable feature in the domain of aircraft conceptual design, and offers the better solution of the two approaches.

In 2004, Lin et al. at the University of Toledo published a paper on the investigation of using an Extensible Markup Language (XML) database approach for multidisciplinary aircraft design. Incorporating data binding and a persistent engine, the authors developed an XML based data model for aircraft design [45]. This model allows design applications to access, manipulate, and manage disciplinary data with an API. However, the problem of how to change the data schema for evolving conceptual design information was not addressed in their work. The data model they proposed seems to be more appropriate for data management at the detail design stage when data structure is fixed, although the implementation example they used is an example of conceptual design.

In light of recent AEE technologies, Kam et al. have developed the Launch Vehicle Language (LVL) for the evaluation of launch vehicle concepts [46]. The evaluation of launch vehicle is inherently a multidisciplinary analysis process. It involves

the analysis of vehicle sizing, trajectory, propulsion, subsystems, operations, cost, and reliability. In their work, the LVL is essentially a common data model based on XML to ensure consistent analysis across the various disciplines involved in the evaluation of launch vehicle concepts.

Li et al, in 2003, proposed an open data management facility called Collaboration Oriented Data Agent (CODA) for distributed multi-agent collaborative work [47]. By combining agent-based technologies with XML technologies, CODA takes the mode of centralized data service and distributed data maintenance. The concept and framework of the ongoing CODA work shows a promising future.

The above research activities in one way or another focus on data structure modeling. In a larger scope, domain modeling has become a very important research topic due to the fact that tools with different domain models cannot communicate with each other because they model the similar things in incompatible ways. In recent years, "ontology" has become a widespread term, drawing research interest in areas such as knowledge engineering and intelligent information integration [48-50]. The aforementioned communication problem is an ontology problem. The purpose of an ontology is for knowledge reuse, interoperability between heterogeneous software applications, and reducing software maintenance costs.

There are many definitions of ontology in the Artificial Intelligence literature. From the view point of information technology, an ontology "*is an engineering product consisting of a specific vocabulary used to describe a part of reality, plus a set of explicit assumptions regarding the intended meaning of that vocabulary, in other words, the*

specification of a conceptualization" [50]. An important concept is that an ontology together with a set of individual instances of classes constitutes a knowledge base. In the medical field, a large amount of research work has been conducted on this research topic. A notable research result is a knowledge based framework and ontology editor named "Protégé" [51]. The Protégé project started with a small application for building knowledge acquisition tools for a few specialized programs in medical planning in 1987. It has been evolved into a Java based platform for knowledge-based systems for research and development. The platform incorporates the open knowledge based connectivity (OKBC) knowledge model, and can interact with standard storage formats such as relational databases and XML. Figure 7 shows how Protégé integrates the components of a knowledge-based system. To connect a particular knowledge base to a particular reusable problem solving method (PSM), users need to map the source and the target class definitions. Marble is a knowledge acquisition tool developed by the Protégé research group which can be used to build the mappings. A generic mapping interpreter applies the mapping to the source knowledge base, and produces a view of the information for the target PSM.

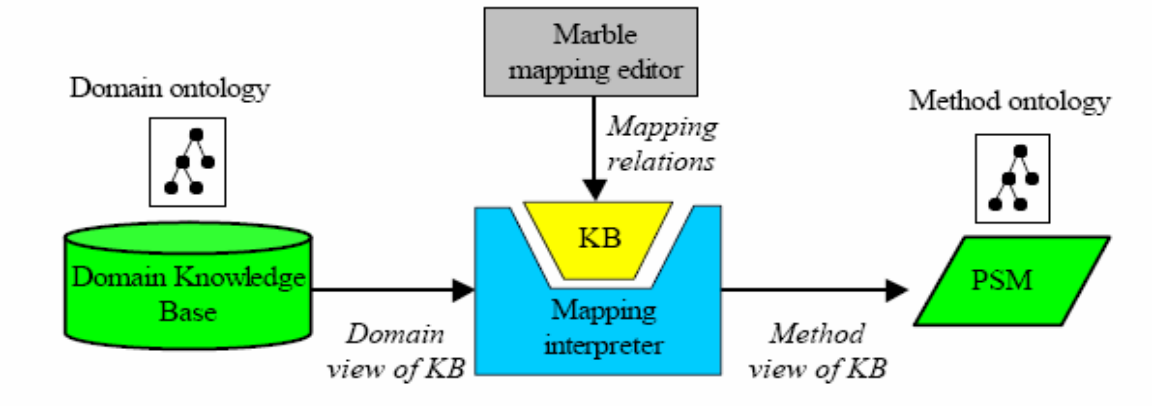

Figure 7. Protégé: Integrating the Components of a Knowledge -Based System [51]

1.3.7 Commercial Integration Software

Integration of application tools is an active research area not only in academia and government, but also in the software industry. On the commercial side, products related to multidisciplinary design and analysis tool integration have been developed and commercialized, though broadly applicable solutions are not available yet. Such commercial endeavors include *AML* (Adaptive Modeling Language) developed by TechnoSoft Inc. [52], *ModelCenter* of Phoenix Integration [53], *FIPER* (Federated Intelligent Product EnviRonment) and *iSight*, both developed by Engineous Software, Inc. [54], etc.

At a high level, the functionalities provided by these software packages are similar. These software packages can be used to link dynamically interacted application computer programs, such as optimization programs, disciplinary analysis tools, and design exploration tools. Used in a design and analysis process together and forming an automated process, these application computer programs can be distributed on different computers. When users execute the design processes, data is automatically passed from one program to another. Compared to traditional manual data translation, this functionality is helpful to reduce errors and save time. The breakthrough enabled by these software packages is that such integration software provides an open platform on which different design processes can be modeled and different analysis tools can be used. Some of them also provide functionality needed to do optimization, Design of Experiments (DOE), visualization of the resulting analyses, and regression analysis such as the response surface method (RSM).

However, using these general-purpose commercial integration software packages, the automation of a design process is accomplished through automatic file processing. File processing is not an efficient way to manage data, especially when the design process becomes very complicated and the amount of design data is huge. Therefore, efficient design data management is a missing capability of these software packages. Some of the software vendors claim that they provide interfaces to other commonly used Product Data Management (PDM) software packages. However, such PDM software packages are usually good for design information management when the data structure is relatively fixed, i.e. after the conceptual design stage.

These integration software packages can be used as vehicles to create an integrated environment for aircraft conceptual design, although the desired effective data management functionality is not provided [21]. In order to concentrate on solving the research problems and save time and effort from the design and development of a distributed computing environment and a graphical user interface (GUI) (the developments of computing technologies do provide us the capability to accomplish this as manifested by these commercial integration software packages), *ModelCenter* is used as an integration tool for the implementation of the research concepts developed in this dissertation. The use of *ModelCenter* facilitates numerous useful tasks, such as debugging, error tracking, and automated optimization/iteration at the current design and prototyping stage of the research. A brief introduction of *ModelCenter* will be provided next. While this particular integration package has proved quite capable, it is likely other integration software would also perform similarly well.

ModelCenter is a commercial product of Phoenix Integration Inc. It is a software package that provides a visual environment for process integration. With a GUI, *ModelCenter* allows users to build up the design process as a series of linked application programs. The application programs used in the design process are connected through wrappers residing on Analysis Servers. Wrappers are software adapters users have to write in order to convert input and output data for the application programs into a standard format that can be linked to other applications. The use of wrappers simplifies the reuse of legacy programs, and the integration of other analysis tools, for example Microsoft Excel. Analysis Server is a Java based software server that allows users to host wrapped application programs for the integrated processes. Security is provided through authentication and encryption. Depicted in Figure 8 and Figure 9 are how the Analysis Server and wrapper capabilities of *ModelCenter* work. To execute design tasks involved in a design process automatically, users can create an execution script similar to a Microsoft Visual Basic code using *ModelCenter*'s scheduling algorithm.

ModelCenter also allows users to conduct trade studies using in-house developed algorithm or tools integrated within *ModelCenter*. These design space exploration tools include tools for parametric studies, optimization, RSM, DOE, and carpet plots. *ModelCenter* also provides Plug-In Toolkit so that users can customize their analysis by directly incorporating in-house trade study algorithms within *ModelCenter*.

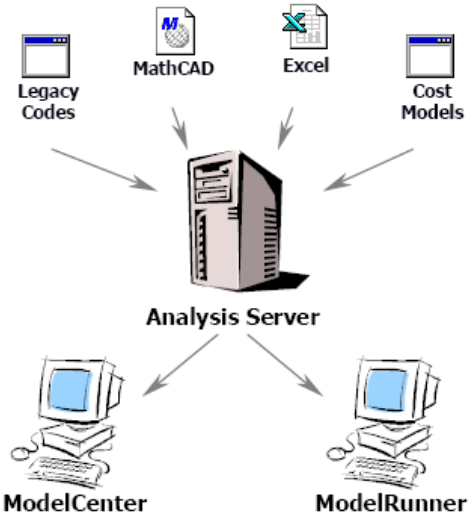

Figure 8. *ModelCenter* Analysis Server [53]

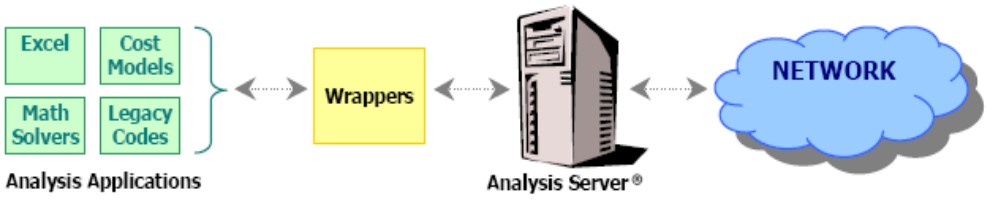

Figure 9. *ModelCenter* Wrapper [53]

CHAPTER 2

RESEARCH QUESTIONS AND HYPOTHESES

From the literature review presented in the previous chapter, the conclusion could be made that we are in need of the next generation aircraft conceptual design environment (NextADE). As has been well recognized, data management has long been a critical issue that needs to be resolved at the forefront of such collaborated design environments. Taking the view that data management should be a fundamental building block of the NextADE, the research reported in this dissertation focuses on finding an effective data management approach in the NextADE from the perspective of aerospace engineering using the state of the art computing technologies. This objective is contingent upon the following two primary research questions and two hypotheses to address them:

- Question #1: *How should we draw the "blueprint" of the NextADE, i.e. considering the building blocks constituting an aircraft conceptual design environment and the desired traits, particularly interoperability, flexibility scalability, and reusability, how should the framework of the NextADE be formulated?*

  The following hypothesis is posed to address this research question based upon the literature research conducted in the previous chapter:

Hypothesis I: *The object-oriented design and analysis approach is an effective approach that can be employed for the design and development of the NextADE.*

Before we address the data management problem for the NextADE, we need to firstly formulate the framework of the NextADE. To meet the current design requirements, the NextADE should be a distributed design environment possessing the ideal traits described in Table 5. Particularly, domain flexibility, analysis scalability, and interoperability are very important focal points. With the hypothesis posed for this research question, which states that instead of building upon a procedural approach such as the state of the practice aircraft conceptual design programs, the object oriented design and analysis approach will be an effective approach for formulating the NextADE. The following tasks need to be addressed in order to answer this research question:

- The object-oriented decomposition of an aircraft conceptual design environment, i.e. identifying the boundaries of the building blocks of the NextADE
- The prototyping, testing, and validation of these building blocks
- The integration of these building blocks to build the initial NextADE

- Question #2: *How can design data be managed effectively in the NextADE, instead of using the traditional file processing approach?*

  Data management is a very important issue that needs to be addressed in a complicated object-oriented design environment. Due to the large amount of data and the increasing complexity of design methodologies, the design process is difficult to model and manage. Effective data management would be able to help to relieve this problem. Once we formulate the framework of the NextADE, we can move on to solve the data management problem. This is the primary focus of the research. Answering this research question can be divided into following tasks:

  - Developing a robust extensible object-oriented data model
  - Implementation of this data model to formulate the aircraft conceptual design data management system using state of the art data management techniques; for example, relational database management and XML data management techniques
  - Integration of this data management system with the object-oriented design environment
  - Experimentation with the design environment supported by the data management system

  The second research question leads to the following hypothesis:

  Hypothesis II: *Effective data management is the key to improving the communication among the heterogeneous design and analysis computer*

programs involved in aircraft conceptual design and thus promoting the efficiency of the overall design process, given that the design methodology (such as sizing algorithms) and the disciplinary analysis computer programs (such as programs of optimization, aerodynamic analysis, propulsion, structure analysis, etc.) needed by a design are available and accessible.

Conquering all angles of these complex questions would be a difficult and time-consuming task. It involves the knowledge in several fields. One person's efforts and knowledge are always limited. However, the first baby step represented by work like this dissertation opens up a whole new world; a little insight will shed more light on related research directions.

CHAPTER 3

AIRCRAFT CONCEPTUAL DESIGN

In Chapter 1, the state of the practice of aircraft conceptual design and research advancements in related areas have been described. An introduction to traditional aircraft conceptual design and the recent developments of complex system design methodologies over the past years, such as Integrated Product and Process Design (IPPD) and probabilistic design methodologies, will be introduced in this chapter. When we design and develop the NextADE, we should not merely reproduce the functionalities we already have with our traditional design tools in a different way. Instead, in light of AEEs, we should have a long-term vision to provide the capabilities that enable designers to conduct the design and analysis of next generation or unconventional concept vehicles. Particularly, we should provide flexibility, scalability, and interoperability to make it easier for incorporating new design and analysis technologies and methodologies.

3.1 Introduction

Aircraft design is an intellectual engineering process of creating a flying machine to meet certain specifications, requirements and/or incorporate innovative, new ideas and technologies [55]. The life cycle of an aircraft can be decomposed into conceptual design, preliminary design, detail design, manufacturing and test, operation and support, and

retirement and disposal. Before a new design is ready to be released to manufacturers, it will have gone through the design process, which includes the stages of conceptual design, preliminary design, and detail design. As we go through these design stages, the level of detail of the design increases. Aircraft conceptual design, as the first stage of designing a new aircraft, plays a very important role in reducing design cycle time and costs. In this design stage, the overall concept of the new aircraft is formed through brainstorming on various alternatives based upon historical data; the high-level structure of the whole complex system is created; and targets on performance, cost, quality, safety, reliability, etc. are set. It is in the conceptual design stage that the basic structure of product information is formed. The entire set of product information will be continually updated based upon this structure throughout the lifecycle of the aircraft. A well-known fact is that more than half of the production cost is committed during the conceptual design stage. However, this early design stage is the cheapest place for making design changes and provides major opportunities to compress design cycle time. In this section, a brief introduction to traditional aircraft conceptual design will be provided.

Each of the aforementioned three design stages, i.e. conceptual design, preliminary design, and detail design, has its own characteristics. Conceptual design is started and constrained by a set of specifications and requirements. It is mainly driven by aerodynamics, propulsion, and flight performance. Other considerations in conceptual design stage include structural analysis, control system analysis, and cost analysis. Traditionally, the analysis in these disciplines is not considered in depth or is not considered at all. However, this tradition is changing. New developments and challenges in the conceptual design stage will be discussed later in this section. Following

conceptual design, the resulting configuration layout is passed to the next design stage – the stage of preliminary design. Traditionally, it is in the preliminary design stage that structural analysis and control system analysis are done in great detail based on the configuration layout. It is also in the preliminary design stage that wind tunnel tests and computational fluid dynamic (CFD) analysis are conducted in order to define the configuration layout precisely. Once the configuration layout is frozen, the design of details, such as the number and placement of nuts and bolts, the sizing of ribs, manufacturing tools, etc., will take place in the stage of detail design. The aircraft can be fabricated after going through these design stages.

Aircraft conceptual design is a process that interprets the requirements into a preliminary layout, and is characterized by creative activities and much change. As will be discussed later, this characteristic is one of the obstacles that must be overcome as we seek effective ways to manage the design information for aircraft conceptual design efficiently, since as change happens the data structure of a design concept usually has to be changed. The goal of conceptual design is to find an acceptable and cost effective aircraft concept to meet the requirement within specified confidence levels. The final aircraft concept is chosen through massive tradeoff studies based on many design alternatives. The general process of aircraft conceptual design is shown in Figure 10. One could also think of this as a special type of systems engineering **[56]** process. This design process is iterative and multidisciplinary in nature. It involves many different aspects from technology selection, geometry configuration, and analyses in several different disciplinary fields or domains (such as aerodynamics, propulsion, cost,

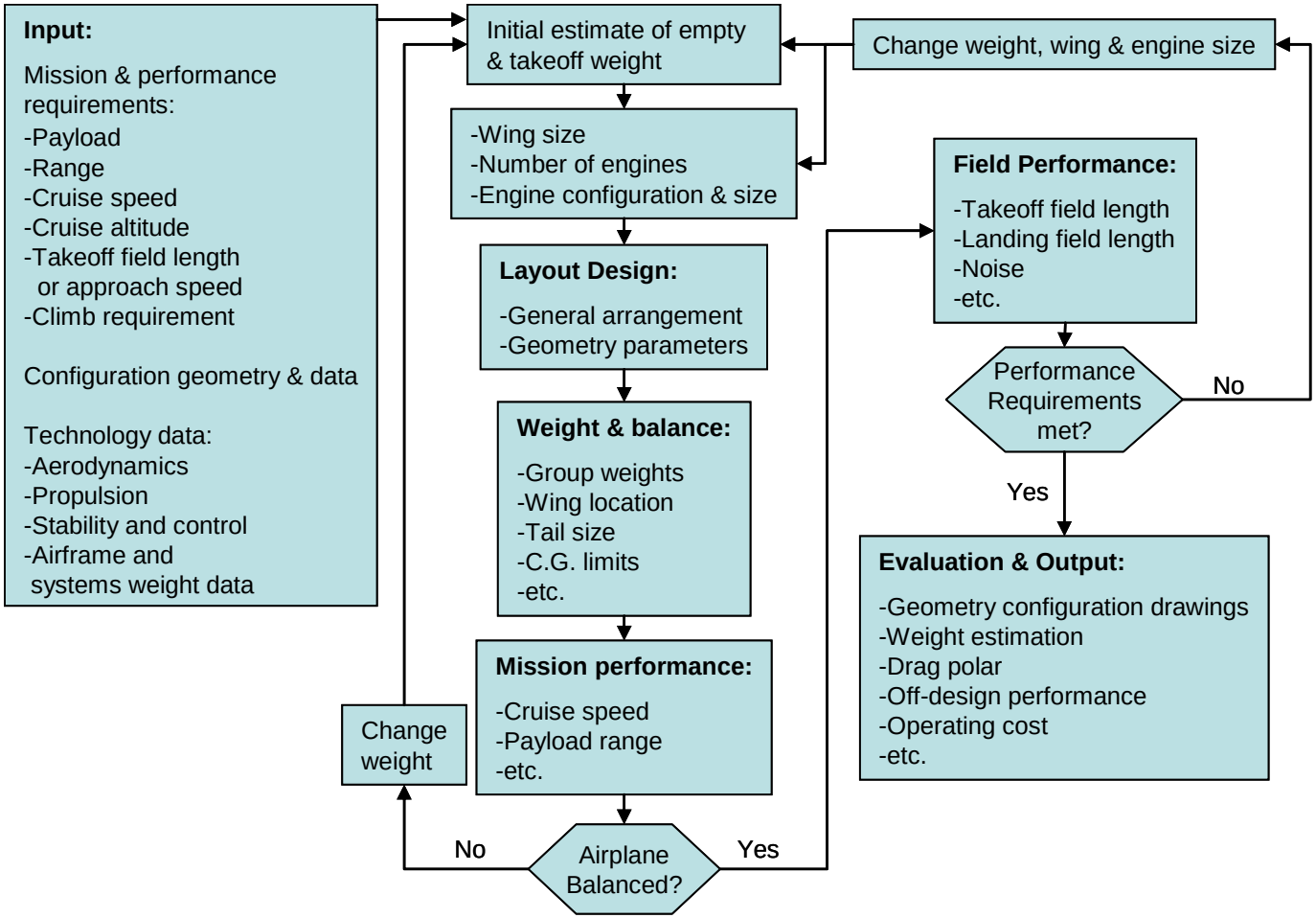

Figure 10. A Sequential Aircraft Conceptual Design Process (Adapted from [8, 57])

performance analysis and optimization), to the requirement tradeoffs. It is hard to reduce this process to a standard one. So it may be more appropriate to say that the steps in Figure 10 are the critical ones that we usually go through in order to design a new aircraft concept.

The critical element of aircraft conceptual design is the sizing process. The sizing process can be described as "*a mathematical algorithm that determines the size and weight of an aircraft based on a specified mission and contributing disciplinary analyses*" [58]. A general traditional sizing environment is shown in Figure 11 [59, 60]. The contributing disciplinary analyses are highly integrated together through common design variables. In order to get the fuel balance and thrust or power balance, the design space that is defined by the ranges of these design variables is examined. The examination is done using multidisciplinary design analysis and multidisciplinary design optimization [59]. The major steps of this examination are constraint analysis and mission analysis. Constraint analysis is about finding a functional relationship between the minimum thrust loading and wing loading at takeoff. Mission analysis is a process to calculate the fuel weight required. The principle of constraint analysis and mission analysis will be described in detail in the next two sections. The given principles are for jet engine-powered aircrafts. For propeller-powered aircrafts, replace "thrust" with "power", and manipulate the equations based upon the relationship between thrust and power.

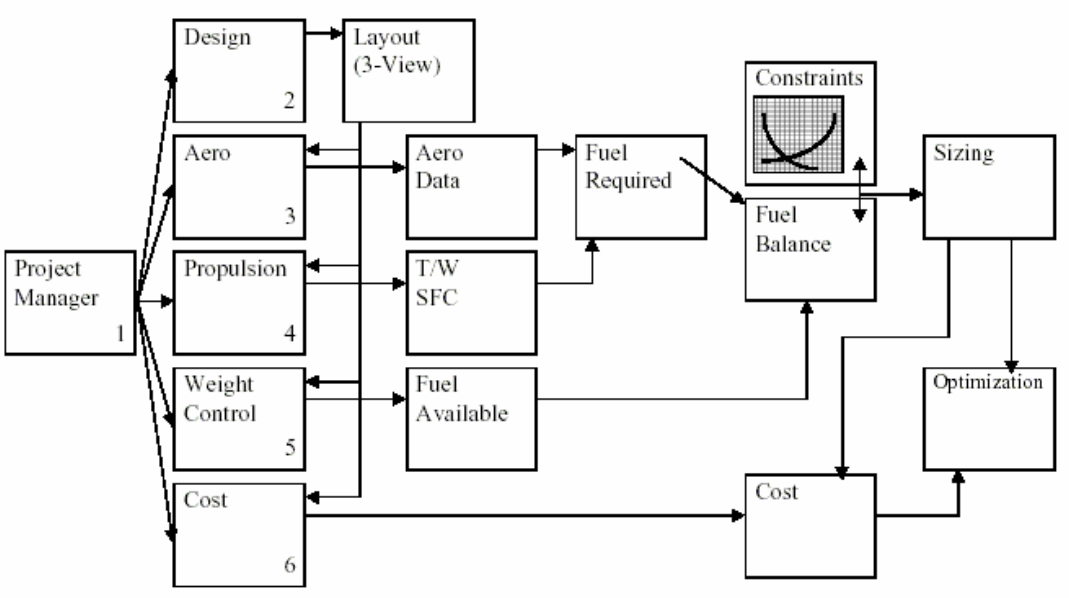

Figure 11. Aerospace Conceptual Sizing/Synthesis and Optimization [59]

3.2 Principles of Constraint Analysis

Thrust loading (T_{SL}/W_{TO}) and wing loading (W_{TO}/S) are the two most important parameters that need to be estimated before we start the initial design layout of an aircraft concept. They are interconnected through a number of critical design requirements and performance calculations, such as takeoff distance, flight required speed, landing distance, and required rate of climb. A typical constraint analysis illustrated in Figure 12 shows the relationship between these two parameters. Once we build up the relationship between them, we can estimate one of them and calculate the other. In Figure 12, any combination

of Thrust loading (T_{SL}/W_{TO}) and wing loading (W_{TO}/S) that falls in the "solution space" meets the constraints considered.

The analytical relationship between thrust loading (T_{SL}/W_{TO}) and wing loading (W_{TO}/S), represented in the working form of equation (1), can be derived from energy consideration [61]. If we treat an aircraft as a moving mass, then its rate of mechanical energy input equals the summation of the storage rate of potential energy and the storage rate of kinetic energy.

$$\frac{T_{SL}}{W_{TO}} = \frac{\beta}{\alpha}\left\{\frac{qS}{\beta W_{To}}\left[K_1\left(\frac{n\beta}{q}\frac{W_{TO}}{S}\right)^2 + K_2\left(\frac{n\beta}{q}\frac{W_{TO}}{S}\right) + C_{D0} + \frac{R}{qS}\right] + \frac{1}{V}\frac{d}{dt}\left(h + \frac{V^2}{2g_0}\right)\right\} \quad (1)$$

where, T_{SL}: installed thrust at sea level

W_{TO}: takeoff gross weight

β: $\frac{W}{W_{TO}}$ and W is the instantaneous weight of the aircraft

α: installed full throttle thrust lapse

q: dynamic pressure ($=\frac{1}{2}\rho V^2$)

n: load factor

S: wing area

K1: coefficient in drag polar ($C_L = K_1 C_L{}^2 + K_2 C_L + C_{D0}$)

K2: coefficient in drag polar

V: velocity

g_0: acceleration of gravity

h: altitude

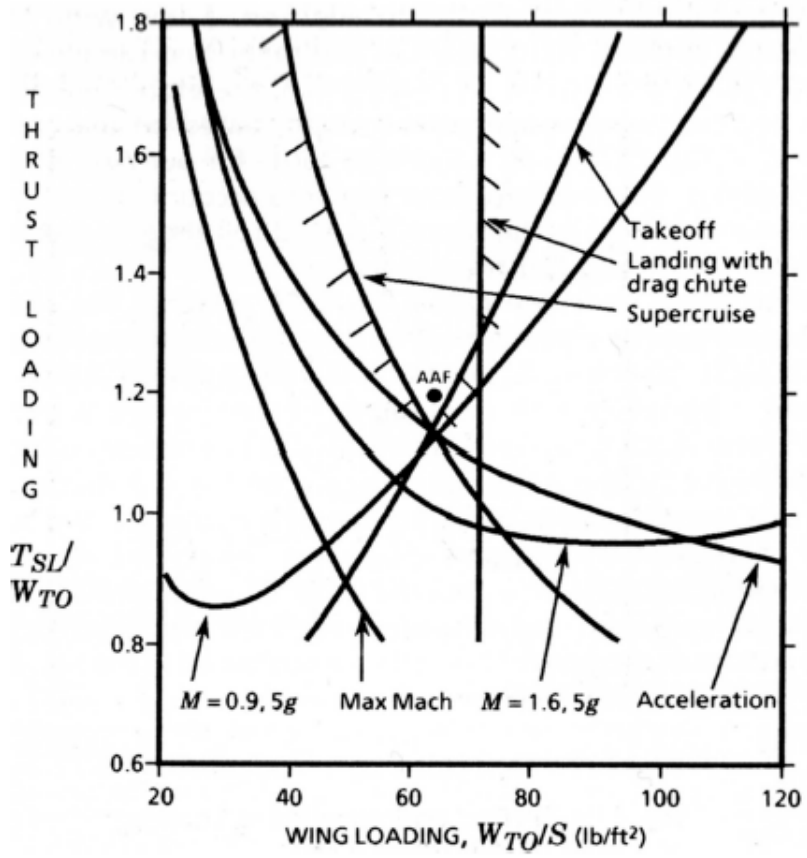

Figure 12. Constraint Analysis: Thrust Loading vs. Wing Loading [41]

The boundaries on the constraint analysis diagram can be calculated base on equation (1). For example, for constant altitude/speed cruise, we have:

$$\frac{dh}{dt} = 0, \quad \frac{dV}{dt} = 0, \quad L = W, \quad and\ R = 0$$

Substitute these relationships into the equation and we would obtain the following relationship between thrust loading (T_{SL}/W_{TO}) and wing loading (W_{TO}/S) with known values of h and V:

$$\frac{T_{SL}}{W_{TO}} = \frac{\beta}{\alpha}\left\{K_1 \frac{\beta}{q}\left(\frac{W_{TO}}{DS}\right) + K_2 + \frac{C_{D0}}{\frac{\beta}{q}\left(\frac{W_{TO}}{S}\right)}\right\} \tag{2}$$

3.3 Principles of Mission Analysis

During the conceptual design stage, the thrust/power balance and fuel weight balance are obtained through constraint and mission analyses. The purpose of mission analysis is to establish the scale of the aircraft through the estimation of takeoff gross weight (W_{TO}). Mission analysis is an important and complicated analysis step. It requires the information from different disciplinary analyses. Or from the viewpoint of designing and developing of the NextADE, in an aircraft sizing environment, mission analysis is the integrator, through which different disciplinary analyses are linked together due to its multidisciplinary nature. It needs to communicate with other disciplinary analysis programs very frequently.

The takeoff gross weight (W_{TO}) is the sum of the required fuel weight (W_F), the payload weight (W_P), and the empty weight of the aircraft (W_E). The payload weight (W_P) includes expendable payload weight and permanent payload weight.

$$W_{TO} = W_F + W_P + W_E \tag{3}$$

In this equation, the payload weight (W_P) is specified in the design requirements and specifications. The empty weight can be calculated statistically based upon historical data [61, 62]. If historical data are not available, we can calculate W_E using weight analysis

tools based upon the geometry configuration of the design, including engines, avionics systems, etc. A large variety of such tools is available. The required fuel weight (W_F) is unknown. However, it can be calculated by flying the aircraft through its entire mission "on paper". We calculate the weight of fuel burned during each mission leg (W_f), for example, warm-up, takeoff, climb, cruise, descend, loiter, landing, etc, and then sum them up for the total mission fuel required.

For the standard case of dissipative fuels, the calculation of the required fuel weight (W_F) is straightforward. The weight of fuel consumed at each mission segment (W_f) can be calculated based on the laws of energy conservation and mass conservation [61]. The rate at which the aircraft weight diminishes due to the consumption of fuel is:

$$\frac{dW}{dt} = -\frac{dW_f}{dt} = -TSFC \times T \tag{4}$$

where, TSFC is the installed engine thrust specific fuel consumption; T is the installed engine thrust.

The rate of energy transformation between mechanical energy input and potential energy and kinetic energy is:

$$(T - D)V = W\frac{dh}{dt} + \frac{W}{g_0}\frac{d}{dt}\left(\frac{V^2}{2}\right) \tag{5}$$

Based on equations (4) and (5), the fuel weight can be calculated for different mission segments. For example, for constant speed/altitude cruise, with given flight conditions, the following formula can be derived – essentially arriving at the Breguet range equation:

$$W_{final} = W_{initial} \exp\left\{ -\frac{TSFC}{V}\left(\frac{C_D}{C_L}\right)\Delta S \right\} \tag{6}$$

where, C_D is the drag coefficient and ΔS is the range.

Equation (2) is an analytical formula. The corresponding approximation formula for numerical calculation is:

$$W_{final} = W_{initial} - TSFC \times T \times \Delta t \tag{7}$$

Equation (7) can be derived directly from equation (4). It can also be derived from equation (6) using the Taylor series expansion of the exponential function. For the purpose of simplicity and saving computing time, equation (7) is commonly used. In this formula, Δt refers to the time to cover the range of cruise.

Another example relates to the mission segment of climb and acceleration. In this case, the final weight can be analytically calculated as:

$$W_{final} = W_{initial} \exp\left\{ -\frac{TSFC \times \Delta(h + V^2 / 2g_0)}{V \times [1 - (C_D / C_L) \times W_{initial} / T]} \right\} \tag{8}$$

The same as the calculation of the cruise segment, equation (7) could also be used for a numerical approximation of this calculation. The difference is the interpretation of Δt. Here it refers to the time that it takes for an aircraft to climb from one energy height to another energy height, where the energy height is defined as:

$$z_e = h + V^2 / 2g_0 \tag{9}$$

3.4 Recent Developments

Along with the increasing complexity of aircraft itself, design methodologies are becoming more and more complicated. The focus of design has been shifted from merely performance to including affordability consideration. Affordability is defined as the ratio of a system's benefits over the cost of achieving those benefits [63]. The development of technologies over the last century has made an aircraft a very complex system to design. In 1903, the Wright brothers turned the dream of flight into reality and changed the world forever. At that time, their aircraft was made of steel, wood, and cloth. They even made the engine themselves. Now a hundred years later, it is unrealistic to expect that several people can build a modern aircraft from scratch in a bicycle workshop. As in the program for Joint Strike Fighter (JSF), the focus of designing such a complex system has been shifted from performance to affordability [64]. From its definition, it can be seen that affordability is a measure of life cycle cost. Life cycle cost is made up of the cost of research, development, test, and evaluation (RDT&E), the cost of production, the cost of operation and support, and the cost of retirement and disposal [65]. Design for affordability implies that in the design decision-making process, we should balance mission capability with other system effectiveness attributes, while keeping cost under close attention.

Traditionally, we used the serial-based system design approach to design complex systems. As a result, design freedom decreases rapidly, knowledge of design increases slowly, and life cycle cost gets locked in too early. Figure 1 of Chapter 1 illustrates the paradigm shift from today's design process to the future goals. We want to bring more

design knowledge to the early design stages while keeping design freedom open longer and shifting to a more gradual cost committed curve. As stated in the NSF strategic planning workshop final report of 1996, “Research Opportunities in Engineering Design”, the most important things to do in the conceptual design stage are to increase design knowledge and maintain design freedom [1].

In order to increase design knowledge and make the right decision during the early design stages, more emphasis should be placed on enhancing systems analysis using modern system engineering approaches, e.g. Concurrent Engineering (CE) and IPPD. CE is a systematic approach to creating a product design that considers all elements of the product life cycle from conception through disposal. CE defines simultaneously the product, its manufacturing processes, and all other required life-cycle processes, such as logistic support. CE is not the arbitrary elimination of a phase of the existing, sequential, feed-forward engineering process, but rather the co-design of all desired downstream characteristics during upstream phases to produce a more robust design that is tolerant of manufacturing and use variation, at less cost than sequential design [66]. IPPD is an outgrowth effort of CE. The Department of Defense (DoD) defines IPPD as "*a management technique that integrates all acquisition activities starting with requirements definition through production, fielding/deployment and operational support in order to optimize the design, manufacturing, business and supportability processes*" [67]. IPPD focuses on the early integration and concurrent application of all the disciplines that play a part throughout a system’s life cycle. The increase of design knowledge at the conceptual design stage also requires high fidelity disciplinary analyses, rather than linear approximations at the conceptual level. These analyses can be

integrated into the core with varying levels of fidelity as depicted in Figure 3. At the top of the hierarchical analysis structure are the geometry design and mission analysis based on guesses, estimates, and historical trends. The next level consists of domain specific tools using first-order methods of low fidelity based on minimal vehicle specification. In order to capture complex phenomenon and increase the accuracy, higher order methods can be used, though they are time consuming. Approximations such as Response Surface Equation (RSE) can be used to integrate higher fidelity modules into the design environment. Today, the distinction between conceptual design and preliminary design tends to be blurred. As a result, the amount of knowledge we have about a design at this design stage keeps increasing.

Next, maintaining design freedom can be achieved through a probabilistic design approach. Probabilistic design finds feasible regions of good design for the design variables. It is important to incorporate robustness into the design process to make the design insensitive to adjustments in the later stages of the design. Unlike a traditional deterministic approach, which searches for optimal or point solutions by assuming that there is no variability in computing the value of the objective function, robust design is a probabilistic design approach, which searches for satisfying solutions to multidisciplinary design problems. During robust design, a designer seeks to determine settings of the control parameters that produce desirable values of the objective function mean, while at the same time minimize the variance of the objective function probability density function [68]. Therefore, robust design is a nondeterministic approach, and is concerned with both the objective function mean and the variability due to uncertainties. Robust design simulation (RDS), illustrated in Figure 13, is a collaborative probabilistic

multidisciplinary approach to aircraft design within the so called CE/IPPD environment [69]. Aircraft conceptual design involves determining an appropriate value for a figure of merit, called the objective function, which is a function of selected design variables and noise variables, such as wing area, thrust-to-weight ratio, turbine inlet temperature, fuel cost, etc. RDS views the chosen objective as a distribution function introduced by noise variables [69]. Unlike design variables, designers can't control the values of noise variables. The distribution type, such as normal distribution, or triangular distribution, must be defined for each of the noise variables. The cumulative effect of all these uncertainties, or distributions, causes the overall variability of the objective function. It is desirable to minimize the dependence of the objective function on noise variables. The core of the RDS consists of vehicle sizing and synthesis combined with an environment to simulate its operation. Information provided to the core includes requirements, physic-based models, process-based models, required new technologies, and uncertainties. In this environment, decision-making is driven by affordability, which is the measure of customer satisfaction. From the viewpoint of the system architecture of this design environment, or from the viewpoint of design itself, it is also important to incorporate flexibility, modularity, adaptability, and reusability to provide additional freedom to adjust and adapt to changes arising later.

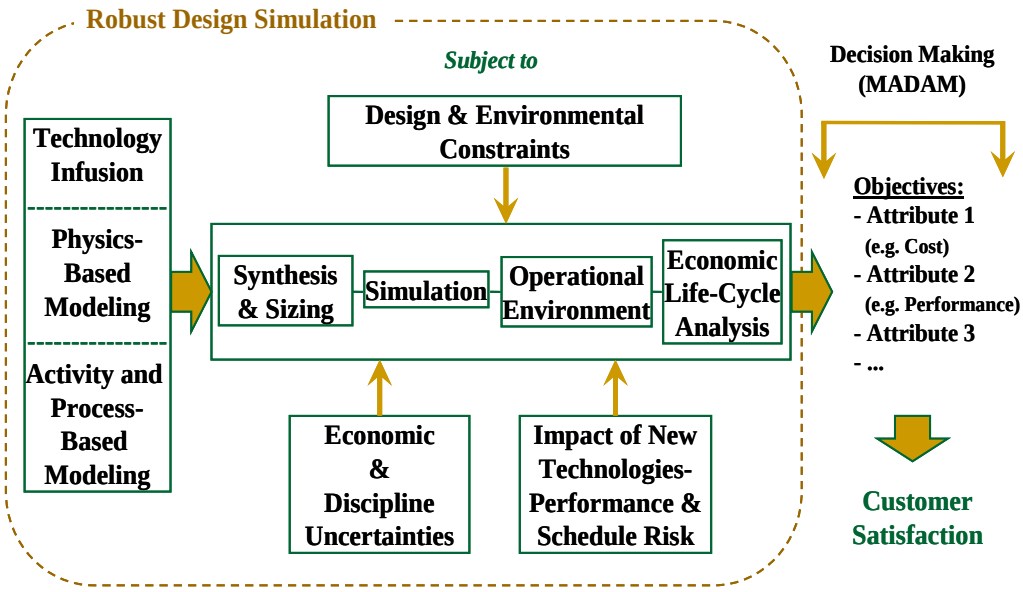

Figure 13. Modeling and Simulation Environment of Robust Design [69]

What are the consequences brought by these new design methodologies in aircraft conceptual design to the management of design information? As shown in Figure 14, the traditional design approach results in an uneven distribution of knowledge and efforts, which concentrate more in the later design stages when making design changes is more expensive. However, the new design methodologies bring more knowledge upfront and make it cheaper to make design changes as shown Figure 15. There is no free lunch. Besides the fact that the early design stage is prolonged, from the viewpoint of design information, the important consequences of using these new design methodologies are that the amount of information generated for a design becomes enormous and that the structure and the flow of the information are more complex. Therefore, an effective data management system must be implemented to manage the design information in order to reduce design cycle time and cost, and improve design quality. In the data management

environment, design and analysis modules should be tightly integrated; the flow of design information with the impact of frequent design changes should be available as quickly as possible. For a collaborative and distributed aircraft conceptual design using the Internet, designers at different locations should be able to share design information that is consistent, in a well understood format, with the minimum amount of redundancy and maximum security.

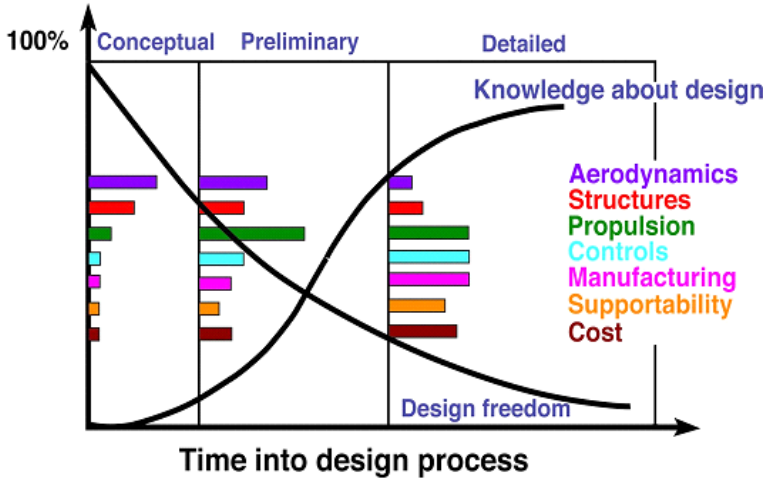

Figure 14. Traditional Uneven Distribution of Knowledge and Efforts

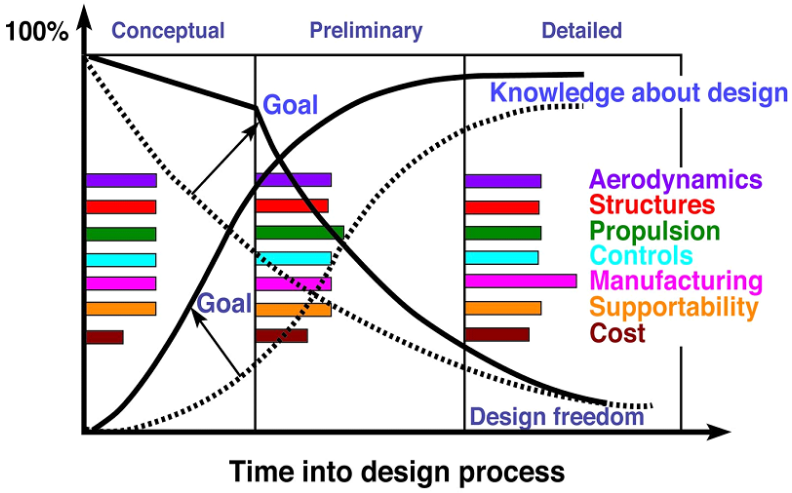

Figure 15. Improved Distribution of Knowledge and Efforts

In summary, the NextADE should not only provide the necessary functionality for both designing conventional and unconventional aircrafts but also provide means to facilitate these new design methodologies. In order to effectively manage the enormous amount of information, the data management issue remains at the forefront and needs to be investigated thoroughly.

CHAPTER 4

AN OVERVIEW OF

ENABLING COMPUTING TECHNOLOGIES

The rapid development of computing technologies, especially information technologies and the object-oriented design and analysis methodologies and techniques, over the last decade is the enabler of designing and developing the NextADE. This chapter serves as a brief overview of relevant computing technologies, including object-oriented methodologies and techniques, data management technologies, the Internet, and the Web.

4.1 The Internet and the World Wide Web

A computer network is a data communications system that links two or more computers and peripheral devices and enables transfer of data between the components in order to share resources, such as hardware, software, and/or data. The Internet is the largest computer network in the world [70]. It is the international collection of computer networks. *"The Federal Networking Council (FNC) agrees that the following language reflects our definition of the term "Internet". "Internet" refers to which - i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the*

Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein" [71]. Through the Internet, computers are connected and able to communicate with each other via many method including email, file transfer, newsgroups, and the World Wide Web. The Internet began as a method to link American military computers in the 1960s. Now, the Internet has become a public computer network linking millions of people around the world and a part of everyday life. The development of the Internet is still ongoing. The wireless mobile Internet (and thus ubiquitous computing) is one of the new paradigms.

The World Wide Web (WWW or Web) is the fastest growing part of the Internet. The Web is a hypertext-based, client/server distributed information system originally created for researchers at CERN, the European Laboratory for Particle Physics in Geneva, Switzerland, to facilitate sharing research information. As said by the inventor of the Web, Tim Berners-Lee, "*the World Wide Web is the universe of network–accessible information, an embodiment of human knowledge"* [72]. It is a system of Internet servers that uses HTTP (Hyper Text Transfer Protocol) to transfer specially formatted documents. The documents are called web pages. The web pages are formatted in language called HTML (Hyper Text Mark-up Language) that supports links to other web pages by their URLs (Uniform Resource Locators). Web pages include text as well as multimedia, such as images, video, sound, etc. The client program, called the web browser, runs on the user's computer and allows user to "surf" the Web.

4.2. Object-Oriented Concepts, Methodologies, and Techniques

Unlike the procedural approach, which is built upon top-down structured algorithm decomposition, the object-oriented approach decomposes a system according to the key abstractions in a problem domain [9]. The latter highlights the agents causing action or the subjects the action operations act upon while the former emphasizes the ordering of events. Each of them has its own advantages and disadvantages. However, the object-oriented approach is more suitable for the design and development of a complex system like the NextADE because it works more effectively for organizing the inherent intricacy of a complex system. The object-oriented approach is employed in this research. The basic object-oriented concepts, modeling techniques, and design methodologies used in this dissertation are introduced in this section.

4.2.1 The Object Model

As briefly introduced in Section 1.2 of Chapter 1, the term of "object" emerged in computer science in the 1970's. Defined by Stefik and Bobrow, objects are "*entities that combine the properties of procedure and data since they perform computations and save local state*" [73]. An object is an instantiation of a class. So, a class is a template that describes the data and the behavior associated with a category of objects, i.e., the instances of the class. Classes are related to one another via inheritance relationships. An object is an entity that has state, behavior, and identity, while a class is defined as a set of objects that share common properties and common behavior. The state of an object is

expressed by the static properties and the dynamic values of each of the properties. The behavior of an object refers to how the object acts and reacts in terms of state changes and message passing. Identity is the property that distinguishes one object from any other object. The object model has four major elements:

- Abstraction: "*An abstraction denotes the essential characteristics of an object that distinguish itself from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer*" [9]
- Encapsulation: "*Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation*" [9]
- Modularity: "*Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules"* [9]
- Hierarchy: "*Hierarchy is the ranking or ordering of abstractions*" [9]

4.2.2 Object-Oriented Programming

Object-oriented programming is built upon the sound engineering foundation of the object model. It is a method of implementation in which programs are organized as cooperative collections of objects. The major advantages of object-oriented programming include:

- Simplicity: through the process of abstraction, complexity of a system is reduced since software objects model real world objects, which appeals to the workings of human cognition. Thus, the software structure is clearer.
- Modifiability: encapsulation or information hiding makes it easier to make minor changes in the data representation or the procedures in an object-oriented program. Changes inside of a class do not affect other parts of a program, since the only public interface that the external world has to a class is through invoking its methods.
- Modularity: a complex system is decomposed into a set of loosely coupled modules where classes and objects are declared.
- Extensibility: adding new features or responding to changing operating environments can typically be solved by introducing a few new objects and modifying some exiting ones.
- Maintainability: objects can be maintained separately, making locating and fixing problems easier.
- Reusability: the object model encourages reuse of both the program and the designs. This leads to the creation of reusable application frameworks.

There are many object-oriented programming languages available, for example, Simula developed by Kristen Nygaard and Ole-Hohan Dahl at the Norwegian Computing Center [74]; Smalltalk developed by Alan Kay, Dan Ingalls, and Adele Goldberg at Xerox PARC [75]; Eiffel by Bertrand Meyer at Interactive Software Engineering [76]; C++ designed by Bjarne Stroustrup [77]; Java by James Gosling [78], Bill Joy, and Guy Steele at Sun Microsystems; C# by Microsoft [79], etc.

Java is used for the implementation in this research. Java is a platform-independent class-based language [80, 81]. The main programming features that the language supports include exceptions, garbage collection, byte code verification, threads, method overloading, packages, and remote method invocation (RMI). It supports multi-levels of implementation hiding, partially abstract classes, final classes, and static variables. In addition to multiple inheritance of "interface", Java provides single inheritance of classes for subtyping and code sharing.

4.2.3 Unified Modeling Language (UML)

In this dissertation, the Unified Modeling Language (UML) is used to express the object-oriented design of the NextADE and the proposed data model. UML is a standard for the creation of models that represent object-oriented software and business systems [82]. The UML specification defines a meta-model that can be used to describe the underlying meaning of each element used in a visual model and the relationships among the elements. Within the UML 2.0 specification [83], there are three types of diagrams used to document various perspectives of a software solution from project inception to installation and maintenance, including behavior diagrams, interaction diagrams, and structure diagrams. Behavior diagrams depict the behavioral features of a system via use case diagrams, activity diagrams, state diagrams, and four types of interaction diagrams. Interaction diagrams are a subset of behavior diagrams used to model object interactions, including interaction diagrams, sequence diagrams, timing diagrams, and communication diagrams. Structure diagrams are used to depict the elements of a specification that are

irrespective of time, including class diagrams, composite diagrams, component diagrams, deployment diagrams, and package diagrams. Following is a brief description of diagrams used in this dissertation. Please see reference [82] for a description of other diagrams.

- The use case diagram is used to show the use cases, actors, and their interrelationships.
- The class diagram shows a collection of static model elements such as classes and types, their contents and relationships.
- The component diagram is used to depict the components that compose a system.
- The sequence diagram models the sequential logic, in effect the time ordering of messages between classifiers.

4.2.4 Systems Modeling Language (SysML)

UML is a powerful modeling tool for software engineering. In order to better serve the systems engineering community so that system engineers can collaborate efficiently with software engineers, in response to the UML for systems engineering RFP (Request for Proposal) of the Object Management Group (OMG), an informal association consisting of industry leaders, software vendors, government, organizations, and academia was organized in 2003. Organizations such as American Systems, BAE Systems, Boeing, Lockheed Martin, Northrop Grumman, Raytheon, DoD, NASA/JPL, NIST, IBM, INCOSE (the International Council on Systems Engineering), and the Georgia Institute of

Technology aligned to define a new domain-specific visual modeling language for systems engineering, which is called Systems Modeling Language (SysML) [84, 85].

By reusing a subset of UML 2.0 diagrams and augmenting them with some new diagrams and constructs appropriate for systems modeling, SysML complements UML 2.0 and supports the specification, analysis, design, verification and validation of complex systems that include components for hardware, software, data, personnel, procedures, and facilities of systems engineering [85, 86]. Common diagrams used by both UML 2.0 and SysML include activities diagrams, block definitions (UML 2.0 classes), internal blocks (UML 2.0 composite structures), sequences diagrams, state machines, and use cases. New diagrams included in SysML are allocations, parametric blocks, and requirements:

- From the viewpoint of systems engineering, allocation is used to describe a design decision that assigns responsibility to meet a requirement or implement a behavior to structural elements of the system. SysML diagrams support systems engineers to use the term allocation to associate behavior and structure in abstract, preliminary, and sometimes tentative ways in order to assess how well the system can be synthesized.
- Parametric constraints specify how a change to the value of one structural property of a system impacts the values of other system structural properties. SysML parametric constraint diagrams complement block diagrams and are used in combination with them to specify aspect such as performance and reliability requirements during system analysis.

- A requirement is a specification of a function that a system must perform or a performance condition that a system must fulfill. SysML provides modeling constructs to represent requirements and to relate them to other modeling elements.

4.2.5 Design Patterns

As a new design paradigm in the context of object-oriented design methodologies, the concept of design patterns emerged and has been explored in recent years. Design patterns are used to specify solution strategies for solving recurring design problems in systematic and general ways [87]. The main idea is to extract the high level interactions between objects and reuse their behaviors from application to application. For example, in the classic design patterns book written by Gamma et al, more than twenty classical design patterns are identified, as shown in Table 6. Based on different purposes, they are classified into three categories: creational patterns, structural patterns, and behavioral patterns. Some of them are applicable to classes, while others are used for objects. It should be noted that the list in the table is not an exhaustive one. Patterns used in this research include the model-viewer-controller (MVC) pattern, the composite pattern, the facade pattern, and the adapter (wrapper) pattern.

Table 6. Design Patterns [87]

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	Factory Method	Adapter	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

- The MVC pattern:

  Figure 16 shows how the MVC pattern works [88]. It can be applied to software architecture design. Using the MVC pattern, we should design the application in terms of three modules:

  - The "Model": It is the core of the application. It maintains the states and data of the application. When changes happen to the model, it should update the views.
  - The "Controller": Users use "Controller" to manipulate the application.
  - The "View": It is the user interface which displays information about the model to the user.

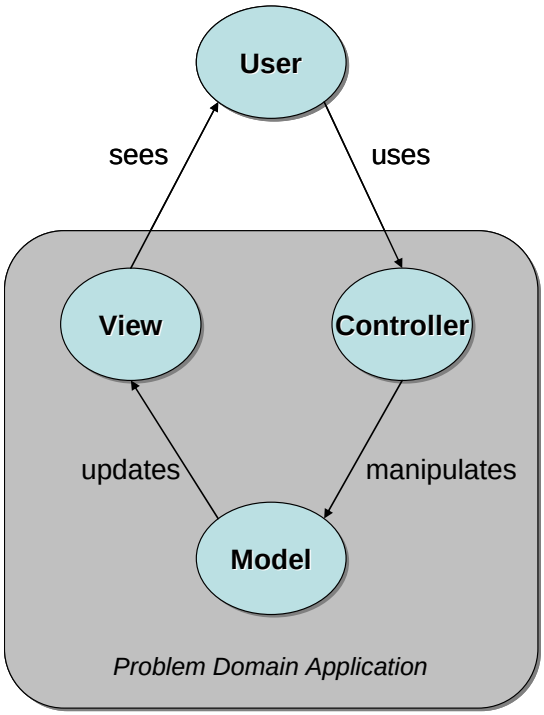

Figure 16. The MVC Design Pattern [88]

- The composite pattern:

  The composite pattern is used to compose objects into tree structures to represent part-whole hierarchical relationships. This way, individual objects and compositions of objects can be treated uniformly by ignoring their differences. Shown in Figure 17 and Figure 18 are the class diagram and a typical composite object structure.

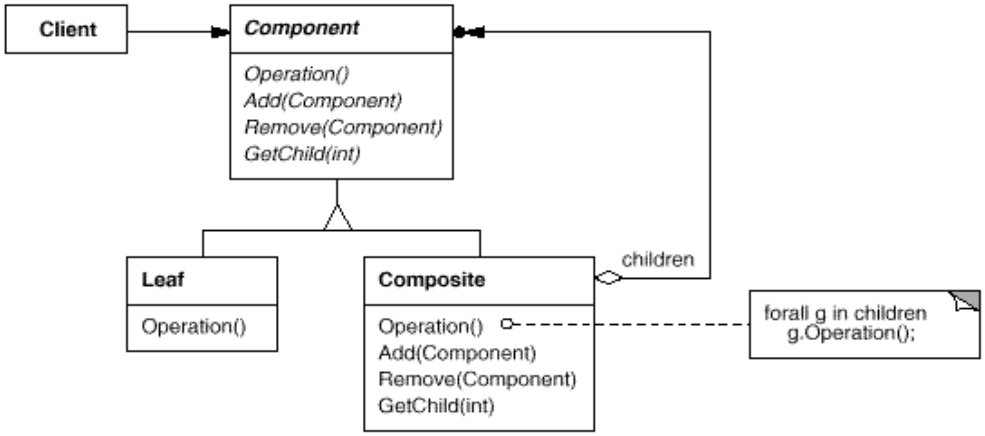

Figure 17. The Composite Pattern Class Diagram [87]

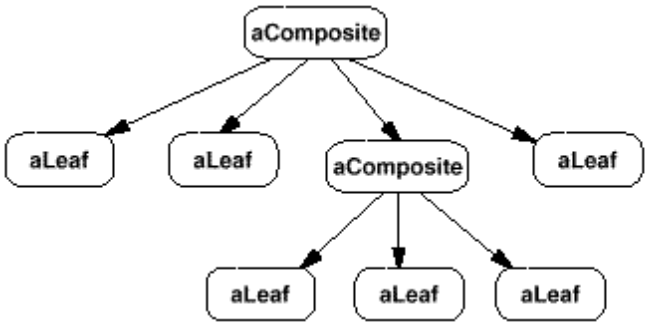

Figure 18. A Typical Composite Object Structure [87]

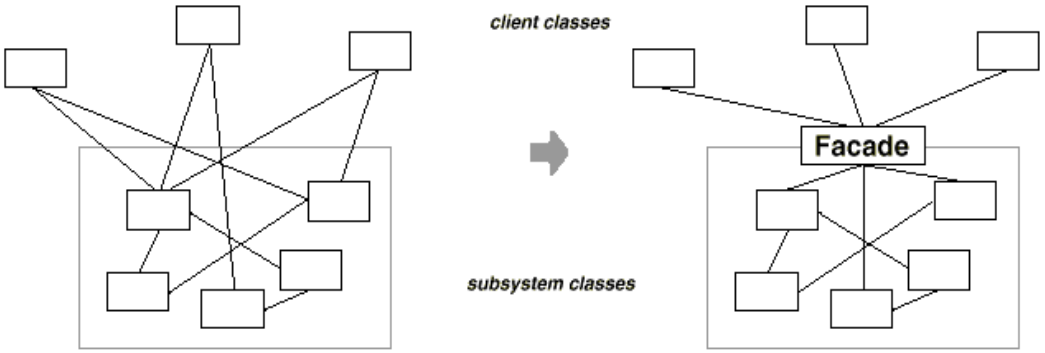

Figure 19. Facade Simplifies the Interface to a Complex Subsystem [87]

- The Facade pattern:

  The facade pattern can make the task of accessing a large number of modules much simpler by providing additional interface layers. It can be used to minimize the communication and dependencies among subsystems as shown in Figure 19. The facade is used to define an entry point to each subsystem level. This pattern simplifies the interfacing that makes large amounts of coupling complex to use and difficult to understand.

- The adapter/wrapper pattern:

  The adapter/wrapper can be used to convert the interface of a class into another interface that users expect so that classes can work together which can't otherwise due to incompatible interfaces. This is an effective and convenient way to reuse existing programs. Figure 20 shows the structure of the adapter/wrapper pattern.

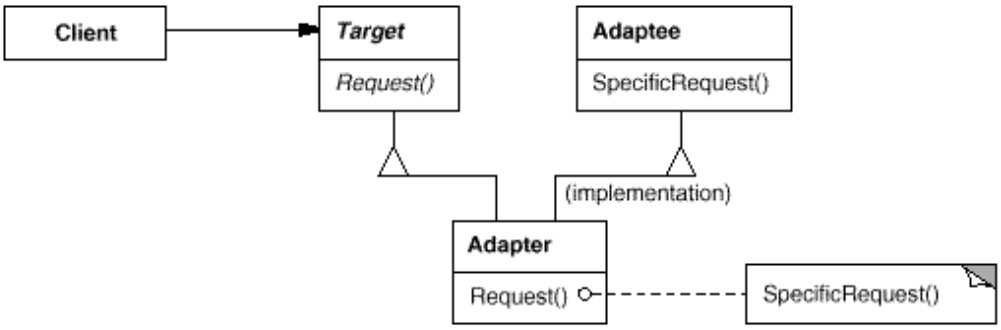

Figure 20. The Structure of the Adapter/Wrapper Pattern [87]

4.3 Data Management Technologies

Provided in this section is a brief introduction of the state of the art data management technologies with a focus on logical database design. A database design methodology can be divided into two stages: logical database design and physical database design [89]. The major difference between these two stages is that logical database design focuses on the "what" whereas physical database design focuses on the "how". At the logical database design stage, the design is mostly independent of implementation details, such as what database management system (DBMS) will be used and what application programs will be involved. The product of this design process is the logical data model and the meta-data that describes the data model. The subsequent physical database stage will take them as the sources of information and tailor them to a specific DBMS. Therefore, at this stage we should know the functionality of the DBMS we use and how

to operate the computer system on which the DBMS resides. In the following sections, basic database concepts, the relational data management approach, the object-oriented data management approach, and the XML data management approach will be introduced briefly. And these are the technologies that will be employed for the design and implementation of the data management system for the NextADE, which will be discussed in the next two chapters.

4.3.1 Basic Concepts

A database is a collection of logically related data stored in a way that it persists and can be manipulated. Persistent means *"continuing without change in function or structure"* [90], i.e. the data stays around after everyone stops working on it and the computer is shut down. A database is a self-describing collection of data that is integrated with a minimum amount of duplication [89]. It is a logically coherent collection of data with inherent meaning and represents some aspect of the real world. The description of the data is called a data dictionary or system catalog, which is the meta-data, i.e. the data about data. Databases provide programs data independence, which means that if new data structures are added to the database, or existing structures in the database are modified, then the application programs that use the database are unaffected. This is an important feature for the NextADE due to the fact that for different designs – or even for the same design – we need to use many different analysis programs. It is usually the case that we can't modify or tailor these programs.

A DBMS is the software that interacts with the users, applications programs, and the database. A schematic diagram of a DBMS is shown in Figure 21. The DBMS allows users to insert, update, delete, and retrieve data from the database. DBMS provides a general inquiry facility to the data stored in the database that is called a query language. For example, SQL (Structured Query Language) is a main query language for relational DBMS.

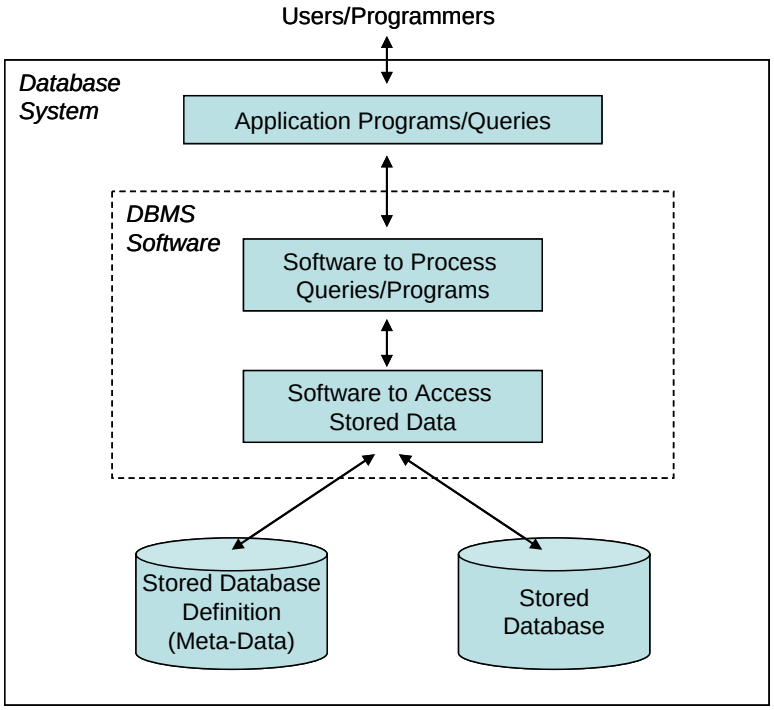

Figure 21. A Simplified Database System Environment (Adapted from [91])

In an engineering design environment, the database approach for data management possesses several benefits compared to the traditional approach of programming with files. In the traditional file processing approach, each user defines and implements the files needed for a specific application as part of programming the application. Because each user might require some data not available from the other user's files, each user maintains separate files and programs. This creates redundancy in defining and storing data that results in wasted storage space and in duplicated efforts to maintain common data up-to-date. In the database approach, a single repository of data is maintained that is defined once and then accessed by different users. According to Elmasri and Navathe [91], following are the main characteristics that distinguish the database approach from the traditional programming with files approach:

- A database system is self-describing. As discussed, the database has meta-data describing the structure of the database. As for the file processing approach, the data structure is defined inside the programs. These programs are constrained to work exclusively with data files confined to the data structures defined in the programs.
- The database approach insulates the program from data and data abstraction. In the traditional file processing approach, the structure of the data files is defined in the programs, so changes to the structure of a file require changes to all the programs that operate on the data file. By contrast, for the database approach, programs can be independent from data since the data structure is defined in a catalog of DBMS that is separate from the programs.

- The database approach supports multiple views of the data. Unlike flat files, a database may have many users, each of whom can require a different view of the database. A view contains a subset of the database and can contain data that is derived from the data explicitly stored in the database.
- The database approach supports the sharing of data and multi-user transaction processing. A multi-user database supports concurrency control to ensure correctness when several users try to update the same data.

For Internet-enabled collaborative design environments, an Internet-based database system, which delivers persistent data via the Internet, is crucial for information management. A Web database system extends the normal definition of a database system to include a Web server and network protocols as shown in Figure 22.

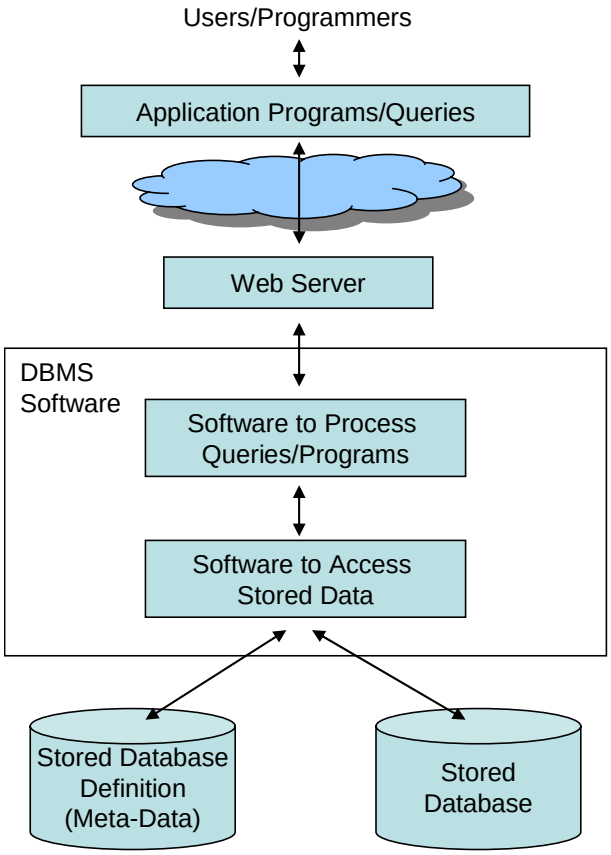

Figure 22. Web Database System (Adapted from [91])

4.3.2 Relational Databases

The Relational Database Management System (RDBMS) is the dominant data processing software in use today. RDBMS is based on the relational data modeling proposed by Dr. E.F. Codd in his paper "A Relational Model of Data for Large Shared Data Banks" in 1970 [92]. A relational database consists of tables that are appropriately structured.

A database is designed based upon a data model. A data model is a conceptual representation of real world objects and events, and their associations. It consists of three components, a structural part, a manipulative part, and a set of integrity rules. A distinguishing characteristic of the relational model is its simple logical structure. In the relational model, all data is logically structured within relations. Relations are used to hold information about the objects being represented in the database. A relation is represented as a table in which the rows correspond to individual records and the columns of the table correspond to attributes. The elements of a relation are called tuples or records in the table. A relational database consists of related tables. In a table, each record must be unique and must be able to be identified through a primary key, which might be a column, or a combination of columns. The relationships among the tables are defined through foreign keys, which is a column or a set of columns within one table that matches the candidate key of some tables.

A popular high-level conceptual relational data model is the Entity-Relational (ER) model. ER modeling is a top-down database design approach. The starting point of ER modeling is the identification of the important data, which are called entities, and the relationships between the data that must be represented in the model. Then more details

are added, such as information that needs to be held within the entities or relationships, which are called attributes, and any constraints on the entities, relationships, and attributes. The ER model has been extended since its introduction in 1976 and leads to the Enhanced-ER (EER) model, which includes advanced modeling concepts such as specialization, generalization, inheritance, and union. Due to the introduction of these concepts, EER can also be used to model object-oriented data relationships.

Another important part of a relational model is its manipulative part. The manipulative part defines the types of operations that are allowed on the data, and a set of integrity rules which ensure that the data is accurate [89]. For a relational data model, entity integrity and referential integrity are two principal rules. Entity integrity refers to a constraint in a base table that no column of a primary key can be null. Referential integrity is a constraint applied to foreign keys – if a foreign key exists in a table, either the foreign key value must match a candidate key value of some record in its home table, or the foreign key value must be wholly null. The types of operations that are allowed on data include updating or retrieving data from the database, and changing the structure of the database. Two main languages exist for accessing relational databases: SQL and Query-by-Example (QBE). SQL has been standardized by the International Standards Organization, and is the standard language for defining and manipulating relational databases.

In order to minimize data redundancy in a relational database, it is important to use normalization techniques to design databases. Normalization is a series of tests on a table to determine whether it satisfies, or violates, the rules for a given normal form. The most commonly used normal forms are the first normal form (1NF), the second normal

form (2NF), and the third normal form (3NF). According to C.J. Date, relational database design should aim at 3NF, not just 2NF and 1NF [93]:

- 1NF: a table in which the intersection of every column and record contains one and only one value.
- 2NF: a table that is already in 1NF and in which the values in each non-primary-key column can be worked out from the values in all the columns that make up the primary key.
- 3NF: a table that is already in 1NF and 2NF, and in which the values in all non-primary-key columns can be worked out from the primary key column, or columns, and no other columns.

4.3.3 Object-Oriented Databases

Relational databases are very good for traditional business database applications. When it comes to more complex data, such as scientific data, relational databases expose certain shortcomings. Scientific data are different than traditional business data. For example, scientific data have more complex structure, new data types, and the need to define nonstandard application-specific operations. Object-oriented databases are the alternative for such complex applications. According to Elmasri and Navathe [91], a key feature of an object-oriented database is that designers can specify both the structure of complex object and the operations that can be applied to these objects without being limited by the data types and query languages available in traditional relational database. What's more

is that an object-oriented database can be directly integrated with the software that is being developed using object-oriented programming languages, while the database becomes a fundamental component of such software.

An Object Oriented Database Management System (OODBMS) is defined as a DBMS that directly supports a data model based on the object-oriented paradigm [94]. The object-oriented paradigm is based on five basic concepts as has been introduced in Section 3.2: 1) each real world entity is modeled by an object; 2) each object has a set of instance attributes and methods; 3) the attribute values represent the object's status; 4) objects sharing the same structure and behavior are grouped into classes; and 5) a class can be a specialized version of one or more classes. From their relationships with object-oriented programming languages, OODBMSs have also adopted concepts such as encapsulation, inheritance, polymorphism, dynamic binding, overloading, overriding, etc. The emphasis of database research has been shifted from relational to semantic and object-oriented models, and only until recently a standard for OODBMS was proposed by the Object Data Management Group (ODMG). The standard consists of the object model, the object definition language (ODL), the object query language (OQL), and the binding to object-oriented programming language.

However, unlike RDBMS, which is supported by a solid theoretical foundation and a very robust infrastructure in terms of the commercial DBMSs, pure ODMSs are presently restrictive as to their modeling power. At this stage in the evolution of database system technology, an emerging class of commercial DBMS called Object-Relational DBMS (ORDBMS) dominates the market. ORDBMS is a good way to enhance the capabilities of RDBMSs with some of the attractive features in ODBMSs in order to deal

with the challenges of complex data management. A similar combination is actually adopted for the implementation of the proposed data management system for the NextADE discussed in the next chapters.

4.3.4 XML and XML Databases

XML, the Extensible Markup Language, is one of the most important developments in document syntax in the history of computing [95, 96]. XML is a W3C (World Wide Web Consortium) endorsed standard for document markup, which defines a generic syntax used to mark up data with simple, human-readable tags. Data included in XML documents are strings of text, and the data is surrounded by text markup that describes the data. A unit of data and its markup is called an element. One of the important features of XML is that it is a meta-markup language, which means that it doesn't have a fixed set of tags and elements, and the language can be extended and adapted to meet different needs. Therefore, developers have the flexibility to define the elements, both data and markups, as needed. Interoperability can be achieved by agreeing to use only certain tag sets, called XML applications, in a particular domain, such as aircraft conceptual design [97]. The markups permitted in an XML application can be defined in a document type definition (DTD) or XML schema. A DTD or XML schema is used to define the data contents and logical constraints on how the XML document should be constructed, much in the same way as data modeling.

XML is useful for data exchange among databases and applications and among multiple applications. Unlike HTML, the markup in XML documents won't tell how the documents should be presented. But the limitation of HTML is that it can't make data as meaningful to computer programs. However, XML can. DTD or XML schema supplies XML files data semantic. The markups in an XML document describe the document's semantics. Thus, XML is self-documenting. It can be used to exchange data among different applications programs (over the Internet) in a way that these programs can understand. These two aspects are very important for scientific data representation and management, since scientific data are usually structured hierarchically and usually have precise meanings only in specific contexts. XML is platform independent, and it delivers portable data. This makes XML implementation relatively robust and future proof eliminating worries arises from the adoption of different hardware and software in the future.

An XML database is a collection of data-oriented XML documents that persist and can be manipulated. The operations provided by the XML database address data manipulation. XML documents can be divided into two categories: data oriented and document oriented. A document oriented document is used when the data are characterized with irregular structure and mixed content, and the physical structure of the data is important. The focus of processing a document-oriented document is the final presentation of the information to the user. A data oriented XML document is used as a data transport when the physical structure of the data is not important. The physical structures of the data are captured by regular structures with many repetitions. The focus of processing a data oriented document is its use and exchange by applications.

Compared to a relational database, the structure of an XML element is more expressive than the relations used in relational databases. A relation in a relational database is an unordered collection of tuples, where each tuple has a fixed set of attributes. The ability to represent the hierarchical structure of XML elements makes it easier to represent more complex data.

As with object-oriented programming languages, XML also has features such as element types, named attributes, and the ability to represent hierarchical structures. One important difference between XML and objects is that objects hide internal structure, while XML exposes all internal structure. An object is a better foundation for programming, but XML is a better foundation for data representation and data exchange although a different language for the manipulation of data is needed.

CHAPTER 5

FORMULATION

In the next three chapters, the two research questions posed in Chapter 2 are answered by finishing their related subtasks. This chapter focuses on the formulation of the proposed solutions to the two research questions. To answer research question #1, *How should we draw the "blueprint" of the NextADE, i.e. considering the building blocks constituting an aircraft conceptual design environment and the desired traits, particularly interoperability, flexibility scalability, and reusability, how should the framework of the NextADE be formulated,* an object-oriented distributed framework for the NextADE is proposed by starting with a discussion on how to object-orientedly decompose an aircraft conceptual design environment. It is then explained why data management in such an environment is a critical problem that needs to be solved at the forefront, and why the traditional file processing approach is not the best solution. Research question #2, *How can design data be managed effectively in the NextADE, instead of using the traditional file processing approach,* is then answered. The answer to this question depends on the solution to the first one. Usually, when designing a database, we go through two design stages: logical design and physical design. During the logical database design stage, the important objects that need to be represented in the database and the relationships among these objects are identified. During the physical design stage, a physical implementation of the logical design will be decided. In this chapter, focusing on the logical design, an extensible, robust, object-oriented data model for aircraft conceptual design data is

proposed for a central integrated data management system, which is used to support the design environment. The implementation of this data model will be presented in the next Chapter.

5.1 An Object-Oriented Distributed Framework Concept

As discussed in Section 1.3.5, the state of the practice aircraft conceptual design programs impose limitations, especially the lack of domain flexibility and analysis scalability, to meet the current design requirements. For example, the requirements of performing variable fidelity aircraft conceptual design and analysis, incorporating the state of the art technologies, and designing unconventional concept vehicles. An additional requirement is that the NextADE should enable designers to do design collaboratively in an extended environment that involves experts, and design and analysis assets at different geographic locations. These problems should be addressed when we design and develop the NextADE. Based upon the characteristics of aircraft conceptual design, and the achievements and lessons learned from the research activities over the last decade, it has been concluded that the object-oriented design and analysis approach, as a good practice, should be exploited. This section will go over the details on the formulation of the proposed object-oriented distributed framework for the NextADE. In the proposed framework, one of the challenges is the modeling of the building block for mission analysis. Unlike the optimization component and other contributing disciplinary analysis components, for which legacy analysis programs can be reused, the mission analysis component needs to be completely redeveloped due to the fact that existing

mission analyses are intertwined closely with specific disciplinary analysis or disciplinary analysis components of a program, i.e. they do not work well with other disciplinary analysis programs. On the other hand, the mission analyses component is one of the most important components, which links the disciplinary analysis tools together. This is one of the reasons why the traditional legacy programs cannot provide designers the flexibility and scalability they need, for example, supporting variable fidelity analyses. The modeling of the mission analysis component will also be particularly addressed in this chapter.

5.1.1 Formulation

The identification of meaningful classes and objects is the key task in object-oriented development. When we design the NextADE, the first question that should be answered is how we are going to decompose an aircraft conceptual design environment object-orientedly, or what should be the classes or objects that make up the key abstraction of an aircraft conceptual design environment. According to Ingalls, "*A system should be built with a minimum set of unchangeable parts; those parts should be as general as possible; and all parts of the system should be held a uniform framework*" [98]. "*If procedures and functions are verbs and pieces of data are nouns, a procedural-oriented program is organized around verbs while an object-oriented program is organized around nouns*" [99], and the verbs are then allocated to the nouns according what those nouns do. How should we describe an aircraft conceptual design environment and what are the nouns? At a very high level, an aircraft conceptual design environment is a computerized simulation environment that enables designers to accomplish the design and analysis activities; and

fundamentally, aircraft conceptual design refers to the process that determines the configuration, the size and weight of an aircraft concept based upon a specified mission through multiple contributing disciplinary analyses. In these descriptions, the nouns are: 1) the "process", 2) the "mission" "analyses", 3) the "contributing disciplinary analyses", 4) the "configuration", and 5) the "size", the "weight", etc. The "size" and the "weight" can be generalized as variables. And the "process" refers to the algorithm of how the "mission" and "contributing disciplinary analyses" should be integrated together.

Therefore, with a high level abstraction, these are the building blocks of an aircraft conceptual design environment. Clarification needs to be made here that these building blocks are conceptual abstractions, and they do not refer to any specific analysis tools or design concepts. If we define the aircraft conceptual design environment as a proper union of design process, analysis tools, and variables, then the design environment, set U, can be represented as:

$$U = P \bigcup C \bigcup A \bigcup V \tag{10}$$

where

U = the aircraft conceptual design environment

P = the set of design processes

C = the set of configurations of different design concepts

V = the set of variables

A = the set of analyses tools = DAs $\bigcup$ MAs, and

DAs = the set of disciplinary analyses tools

= *{Optimization, Aerodynamic, Propulsion, Weight,*

Stability & Control, Cost,}

MAs = the set of mission analyses tools

= *{Warmup, Takeoff, Climb, Cruise, Descend, Loiter,*

Landing, Hover, ...}

In the above representation, "*Analyses*" is defined as the union of "*Disciplinary Analysis*" and "*Mission Analysis*". In turn, "*Disciplinary Analyses*" and "*Mission Analyses*" are infinite sets theoretically although it is not the case in practice. There can be as many disciplinary and mission analyses as designers need. Using the above definition, depending on different design concepts, each element in set P, set C, set V, and set A could be decomposed further, while the elements in set V are atomic. For a specific concept, set P defines the logical relationships among different elements in set C and set A through the elements in set V, and elements in set C and set A are related to and also represented through the elements in set V.

The diagram shown in Figure 23 is a symbolic view of the proposed object-oriented decomposition of an aircraft conceptual design, which is the distributed object-oriented framework of the NextADE without the support of a data management system [21]. For a specific design, we generally know the types of analyses that we are going to perform using certain application tools, and the overall mission we need to work on for a specific concept. It must be pointed out that the analyses objects, i.e. the mission analysis objects and the contributing disciplinary analysis objects, involved in the design environment as shown in Figure 23 and Figure 24 are for illustration purposes only. The actual analyses involved in a real design are more complicated than those shown in these figures.

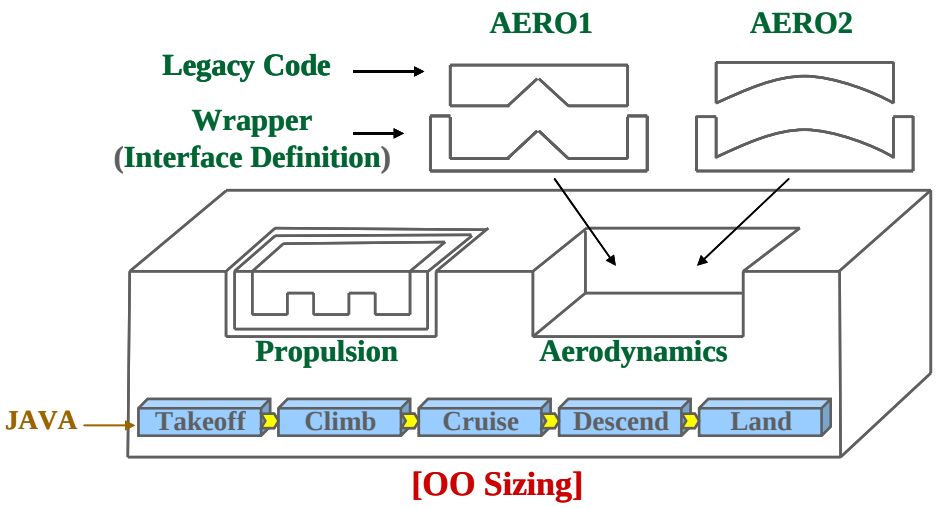

Figure 23. An Object-Oriented Aircraft Conceptual Design Framework: A Tactical View

In this framework, the NextADE is built on a component basis. According to Szyperski, a component is a unit of independent deployment which needs to be well separated from its environment and from other components; a component is a unit of third-party composition which needs to encapsulate its implementation and interact with its environment through well defined interfaces; and a component has no persistent state which means that a component cannot be distinguished from copies of itself [100]. Usually, a component is a heavy weight unit with one instance in a system. It consists of a set of classes or immutable prototype objects.

The whole design environment can be decomposed into components of mission analyses, contributing disciplinary analyses, and variables for a specific design concept, without considering the design process. Again, the design algorithm is reflected in how

these analyses are organized together – the design process. In a design environment, these components come to life through objects, such as contributing disciplinary analysis objects, mission analysis objects, and variables objects. As mentioned previously, at the sublevel, the mission analysis component is decomposed into elementary mission segment objects based upon the mission profile; for example, mission segment objects of warmup, takeoff, climb, cruise, descend, loiter, and landing. The object-based construction of a mission profile makes it possible to generate any mission by combining these atomic mission segment objects in an according order. The contributing disciplinary analyses (CDAs) component also consists of objects, such as optimization objects, aerodynamic analysis objects, propulsion analysis objects, cost analysis objects, etc.

These objects can be wrapped legacy analyses programs, which reside on different computers, or we can write new classes for new design and analysis requirements. The wrappers provide appropriate interfaces to the backbone framework structure. For example, as shown in Figure 23, the aerodynamic analyses programs *AERO1* and *AERO2* have different interfaces. Different wrappers can be written for each of them in order to integrate them with other application programs. Generally, for the CDAs, wrappers are created specifically for each one so that they can be seamlessly plugged into the framework. Future work needs to be done on designing and developing a universal interface that helps to create the wrappers for different application programs. Using the wrappers, a variety of analysis tools can be fully utilized, so instead of being limited to the built-in CDAs, designers would have the freedom to choose particular CDAs according to their needs for either low fidelity or high fidelity design and analyses. Obviously, accompanying this benefit is the drawback of writing wrappers for the chosen

CDAs. However, these wrappers are generally reusable. A library can be created in order to manage these wrappers for reusing.

The proposed object-oriented framework can be viewed more strategically from a multidisciplinary analyses perspective. For example, the overall program structure, and thus the optimization/iteration problem, can be represented using a Design Structure Matrix (DSM) as shown in Figure 24. Since the aircraft sizing process is a highly coupled, iterative process, there can be many feedforward and feedback relationships. Again, in Figure 24, the CDA objects, and the mission segment objects, are shown for illustration purposes only. Ideally, the object-oriented aircraft conceptual design environment is built upon three major "libraries" including the sizing algorithm library (providing domain flexibility), the CDA library (providing analysis scalability), and the mission segments library. These libraries make up the knowledge foundation of the design environment. As can be seen in the magnified "Mission" module, depending on the problem at hand, each component can be divided into sub-level analysis. It can also be treated as a black box as the "Aero" component. The following section illustrates the treatment of the mission analysis component in detail.

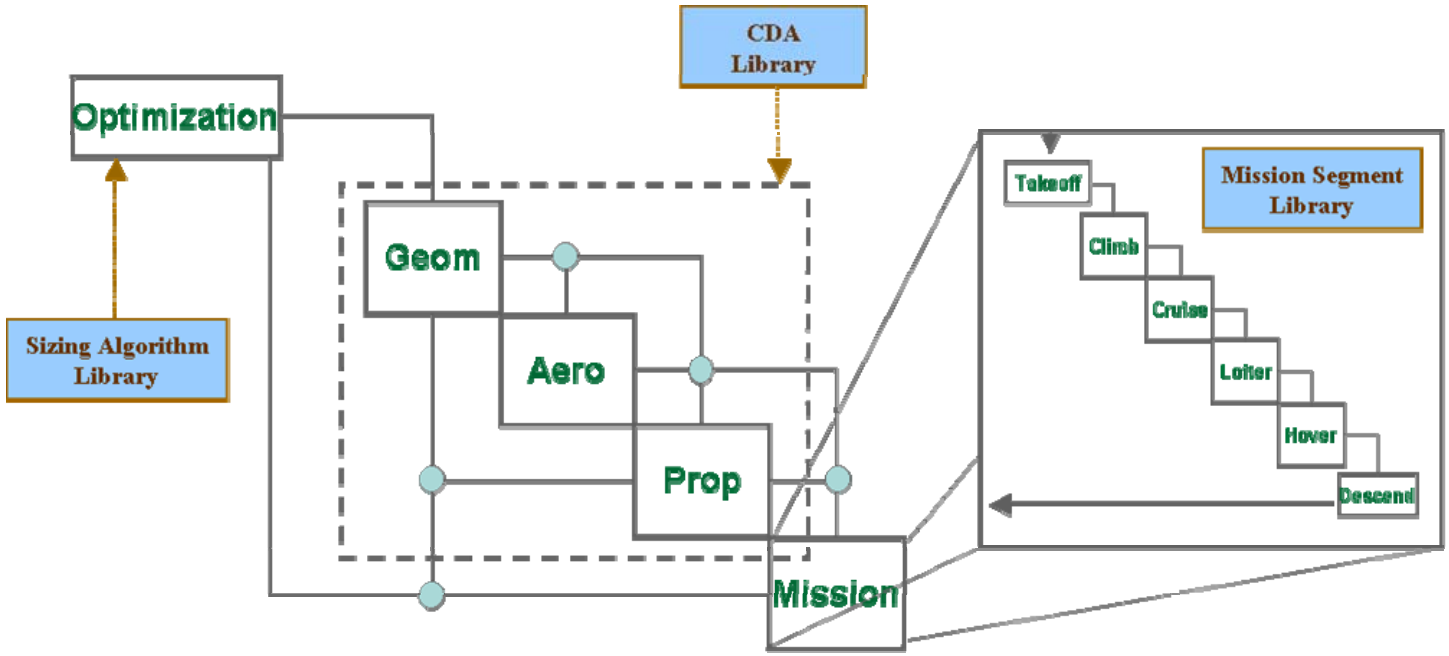

Figure 24. An Object-Oriented Aircraft Conceptual Design Framework without Database Support: a Strategic View Using Simplified DSM

5.1.2 The Mission Analysis Component

The mission analysis component is one of the key components that must be developed for the NextADE. The mission analysis component acts as an "integrator". It needs the support of most of the CDAs, and interacts with them frequently. Experience shows that this is actually one of the most computing-intensive components. However, unlike other components that can be built by reusing existing resources, for the mission analysis component, we have to start from scratch. Existing mission analysis functionality built in specific aircraft conceptual design programs only work well together with those specific disciplinary analyses functions within the same specific aircraft conceptual design programs. For the NextADE, we need to build the mission analysis component such that it is flexible or general enough to be used with different disciplinary analysis tools. The object-oriented decomposition of the mission analysis component will be discussed next.

Conceptually, object-oriented modeling of a mission profile decomposes it into independent, elementary or atomic mission objects. A library containing many different kinds of such elementary mission objects for different types of aircraft can be constructed. With such a library in hand, designers are empowered with the flexibility to construct virtually any mission profile by arranging selected elementary mission objects in an admissible order. The flexibility to add new elementary mission objects into the library when needed, without changing existing ones, is also typically possible.

From an implementation point of view, object oriented modeling of a mission profile involves modeling these conceptual objects in computer programs. The benefits of object oriented programming in this context are encapsulation, inheritance and

polymorphism. Encapsulation keeps data and implementation within the objects – localizing the knowledge within the mission segment objects. Designers can view the elementary mission segment object as a black box that provides services. Inheritance enables the extension of existing mission segment objects without changing them – a class can automatically inherit the variables and methods defined in its superclass. These variables and methods can also be modified in the subclass without affecting the behaviors of the superclasses. Polymorphism enables mission segment objects to act depending on their run-time type – all objects in the same inheritance hierarchy could respond to the requests that their predecessors could respond to. In a practical sense, these traits allow for useful things such as extendable program development, protection of proprietary data, etc.

When decomposing a mission into elementary or atomic objects, two things need to be carefully considered: in order for the objects to be reusable for different mission profiles, the conceptual boundaries and the relationships of these objects. For these elementary mission objects, the finer the modeling, the higher the level of flexibility and reusability. Most importantly they should be able to be used in a continuous time manner and properly calculate fuel consumed for a specific mission segment. Their interfaces (input/output) should be reasonable, easy to use, and provide necessary information to designers.

Accounting for all of these considerations, a mission profile is decomposed as illustrated in Figure 25. This model is extensible since it is object-oriented. *MissionSegment* is a generalized superclass. It contains common attributes and methods of its subclasses: *warm-up, takeoff, climb, acceleration,* etc. *Takeoff* consists of takeoff

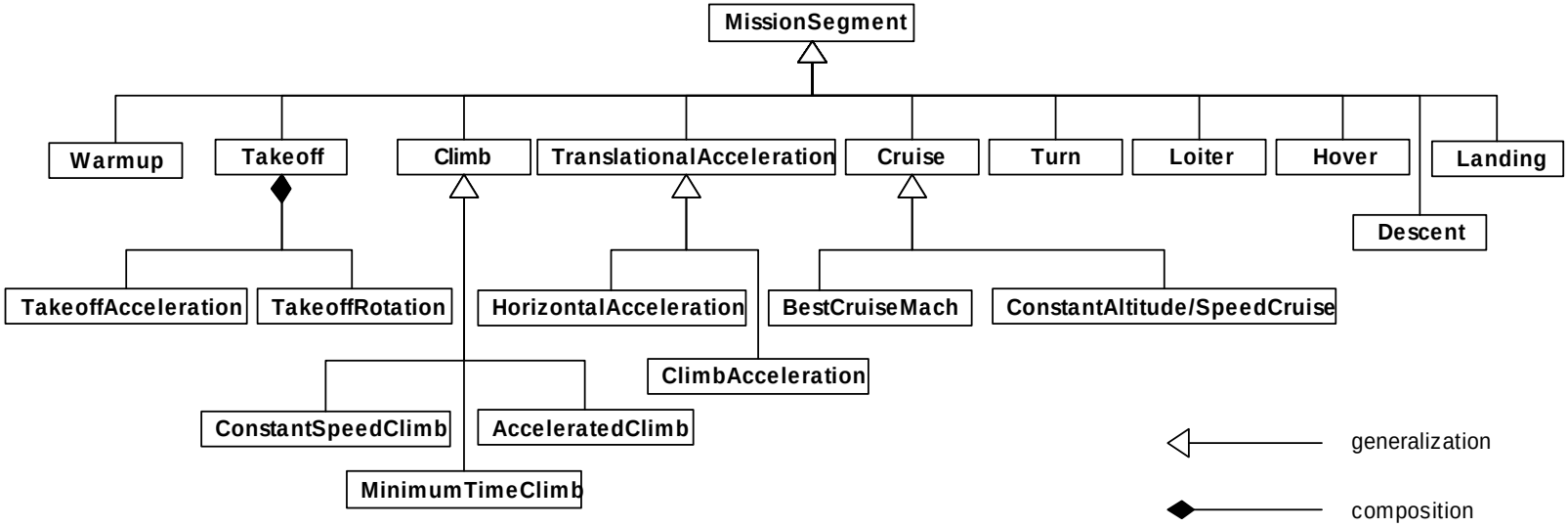

Figure 25. Object-Oriented Decomposition of the Mission Analysis Component

acceleration and takeoff rotation. *Climb* could be further categorized into constant speed climb, accelerated climb, and minimum-time climb. *TranslationalAcceleration* is categorized into horizontal acceleration and climb acceleration. Here, climb acceleration and accelerated climb are implemented using the same class since they are essentially the same. *Cruise* is also divided into two categories, constant altitude/speed cruise and cruise with best cruise Mach number.

5.2 Data Management

In the previous section, the proposed object-oriented framework for the NextADE has been discussed. The question now is why, in such an environment, data management is a critical issue that needs to be addressed. Can we just use the traditional file processing approach as we do in current practice? Unlike traditional aircraft conceptual design programs, the building blocks of this design environment are standalone entities. The design environment cannot be integrated until information can be exchanged freely among these building blocks. Figure 26 shows that the traditional file processing approach for information management in such an environment is not the best solution. As will be discussed later, this approach is a point-to-point translation approach, which is not efficient enough when many analysis tools are involved. What is shown in this figure is actually an abstraction of a real design. It is a mirror image or an abstraction of the data exchange operations of a true analysis process involved in a simple design – the design example used for the first case study conducted in Chapter 7. For this design, the four

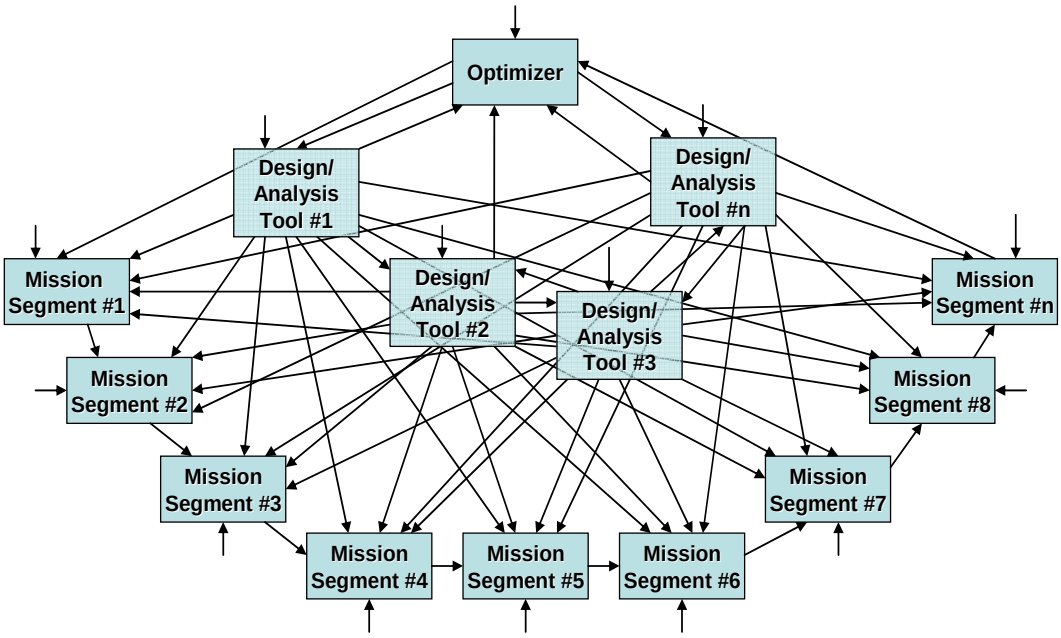

Figure 26. Data Exchange Using the Traditional File Processing Approach

design and analysis tools correspond to geometry design, aerodynamic analysis, atmosphere simulation, and propulsion simulation; the nine mission segments are warmup, takeoff, climb, cruise, turn, cruise, descend, loiter, and landing. As the system becomes larger and more complex, the amount of data operation exchanges will increase rapidly.

There are mainly three approaches for information sharing: shared services via point-to-point translation, neutral interchange format, and neutral authoring [101]. Each of them has its own advantages and disadvantages:

- Shared services via point-to-point translation

  As shown in Figure 27, in order to integrate Application 1 with Application 2, a translator is required because the concepts and terms used by Application 1 are different than those used by Application 2. If Application 1 needs the service provided by Application 2, the request for information must be translated from Application 1 to Application 2, and the response from Application 2 must be translated back into the concepts and terms understood by Application 1. Most of the commercial integration software packages described in Section 1.3.7 use this approach in order to integrate heterogeneous application tools. Shown previously, in Figure 26, is a real-world scenario. The drawbacks of this approach lie in i) the maintenance of the information sharing link, for example, if the terms of Application 1 are changed, then the translator must be updated; ii) direct run-time sharing depends more on network services; iii) much effort is need to build the translators, especially when there is not a direct mapping between the terms of one application to the terms of other applications.

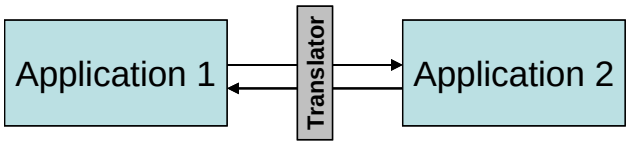

Figure 27. Shared Services via Point-to-Point Translation (Adapted from [102])

- Neutral interchange format

  When there are many applications, typically more than four applications, which need to be integrated, it will be more beneficial to use a neutral interchange format as shown in Figure 28. The neutral interchange format is an intermediate representation language which needs to be designed in a sufficiently expressive neutral format and should support two-way translation between the neutral format and each target application format. The benefits of this approach include i) there is no need for application builders to learn new languages of other applications; ii) the fact that the applications involved in a system are independent with each other; iii) fewer translators, $O(n)$ where n is the number applications, needs to be built compared to the shared services via point-to-point approach, which needs to build $O(n^2)$ translators.

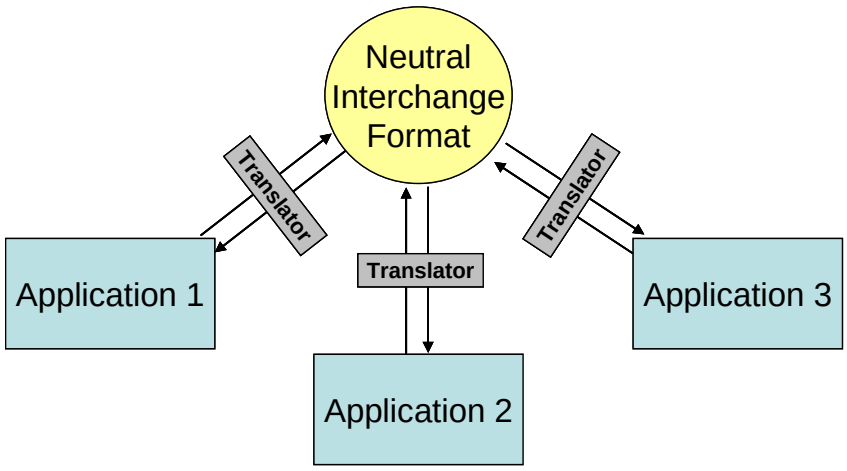

Figure 28. Neutral Interchange Format (Adapted from [102])

- Neutral authoring

  This approach of information sharing is similar to the neutral interchange format approach. However, the neutral format in this approach is used for authoring rather than interchanging as in the neutral interchange format. This means only one-way translation is needed, which is from the neutral authoring format to the target applications, as shown in Figure 29. Compared to the other two approaches, using this approach, i) only one version of knowledge needs to be maintained; ii) the dependence on particular vendor formats is eliminated; iii) it results in an apparently cheaper and more effective retention of information assets although it takes effort to define and maintain the neutral format and interfaces. When using this approach, the difficulty lies in the design of the neutral authoring format. Only within certain limited domains, can this be accomplished.

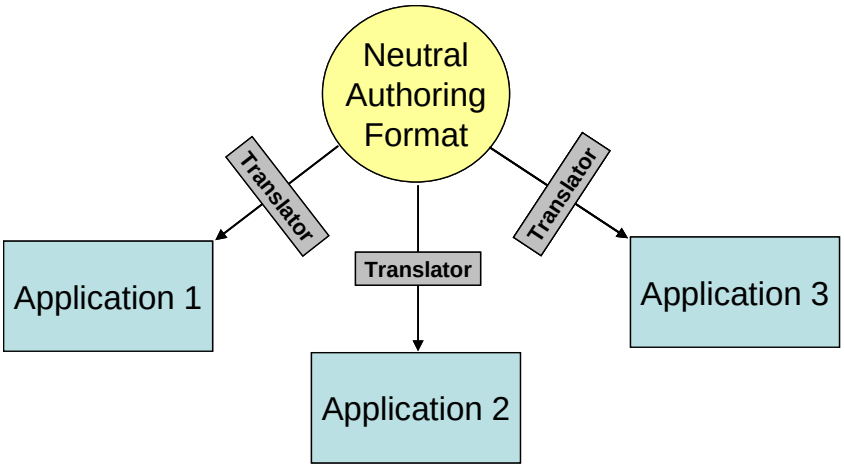

Figure 29. Neutral Authoring (Adapted from [102])

Comparing the pros and cons of the previous three data sharing approaches, one viable solution to the data management problem in the NextADE is the addition of another building block - a supporting central integrated data management system, as shown in Figure 30. The design data will be stored in a neutral interchange format that can be translated amongst target applications. As will be shown later, this is approach is more efficient than how the general-purpose integration software packages integrate different applications programs, which facilitate the shared services via point-to-point approach. By isolating the other building blocks from each other, this approach will also have a significant positive effect on improving the flexibility and scalability of the NextADE.

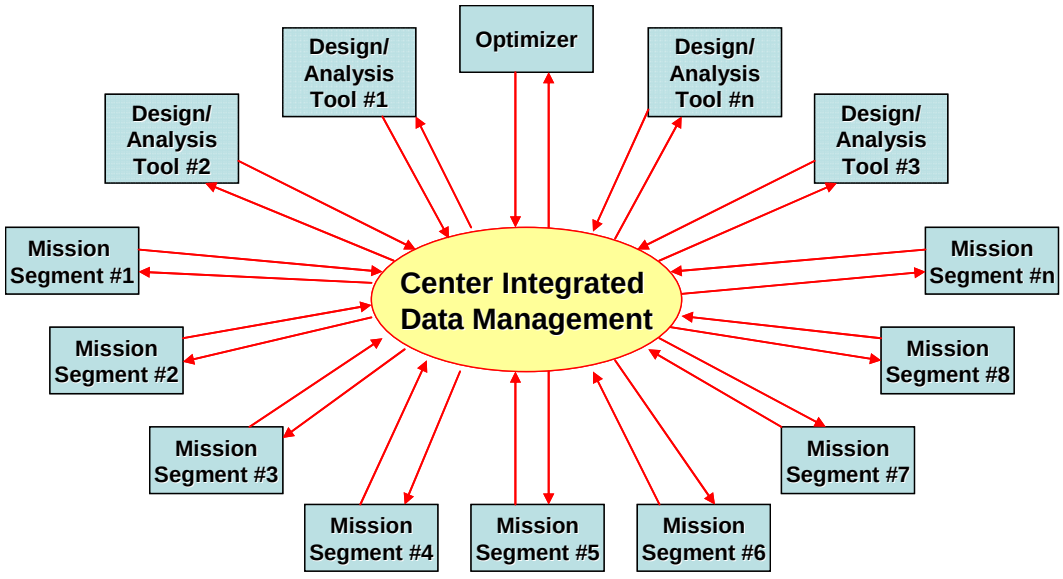

Figure 30. Data Exchange Using a Central Integrated Data Management System

Shown in Figure 31 is a strategic view using DSM of the design environment supported by such a data management system. In this design environment, the objects of the optimization analysis, the contributing disciplinary analyses, and the mission analyses are isolated amongst each other. There is no communication among these objects. They only communicate with the data management system by querying the design information they need for their specific analysis from and storing the analysis results into the data management system. However, as identified previously, it is difficult to design a neutral interchange format for the application programs. Now, there is a problem of how we should build the central integrated data management system. An extensible, robust object-oriented data model is proposed in order to answer this question. It will be discussed in detail in the next section.

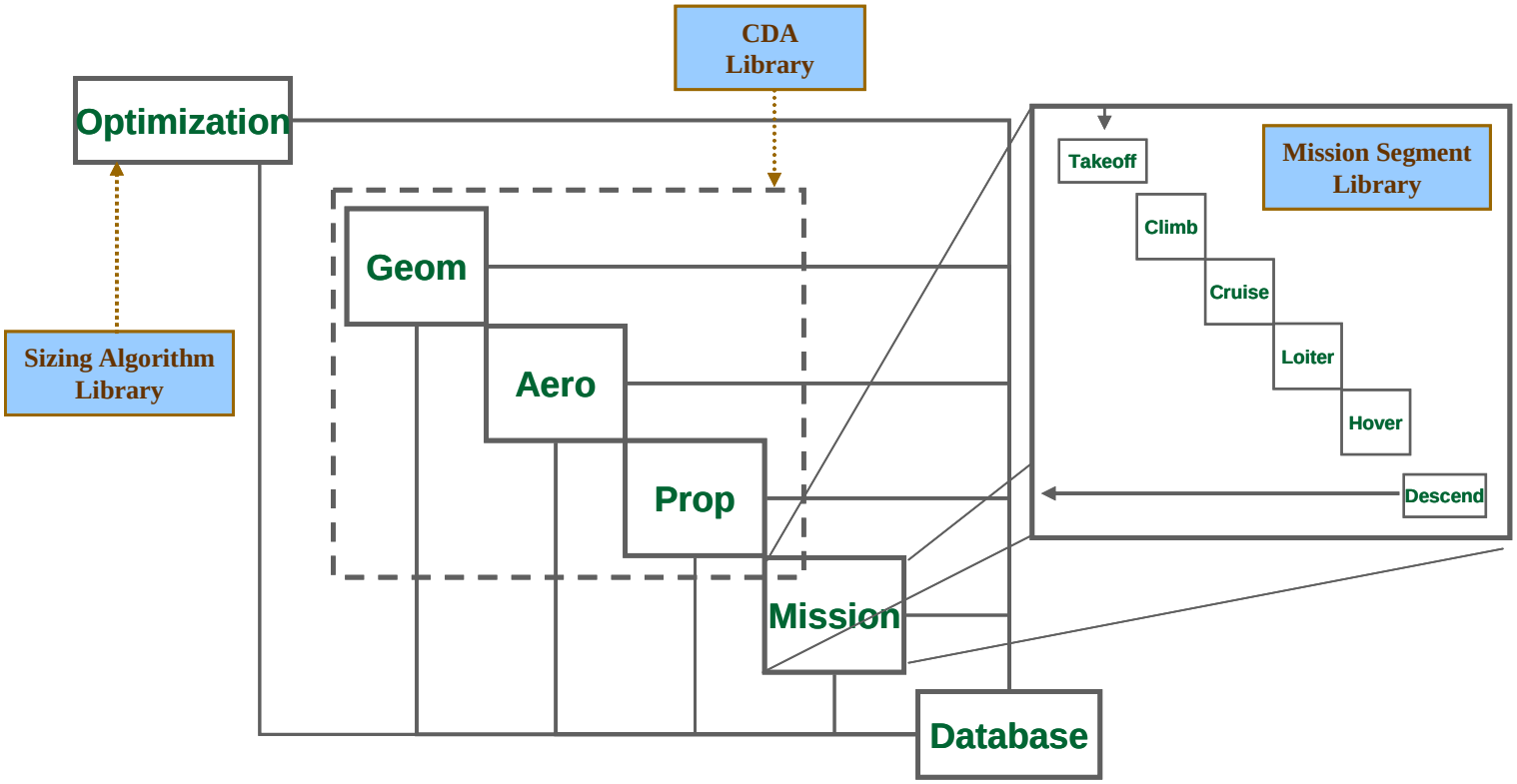

Figure 31. The Integrated Design Environment with the Support of a Data Management System: a Strategic View Using Simplified DSM

5.3 An Object-Oriented Data Model

Based upon the proposed object-oriented framework for the NextADE, aircraft conceptual design data in this environment are modeled as in Figure 32 using UML. This is a project-based, object-oriented data model. Shown in this figure are the classes, their important attributes, and the relationships among them. Each block in the figure represents one class with class name placed at the top of the block, attributes listed in the middle, and methods at the bottom. A string is assigned to each object as a unique identifier – the "id" attribute of each class. In the implementation, Universal Unique Identification (UUID) is used as the "id" of an object. Through the mechanism of combining the hardware addresses, time stamps, and random numbers, UUID makes a data object uniquely identified [103]. It should be mentioned that due to the limit of space, and for clarity, trivial methods operating on the attributes of each class are suppressed in this figure. Major operations, except those mutators/accessors methods of each attribute, will be presented when each data class is discussed.

In the figure, the lines connecting one block with another represent the relationships between the two classes. A line with a diamond arrowhead represents an aggregation relationship. For example, class "GeometryConfiguration" is an aggregate class and an object of class "Component" is a part of an object of "GeometryConfiguration". A line with a triangular arrowhead represents an inheritance relationship. For example, class "Analysis" is a base class and class "Optimization" is one of its derived classes. The numbers associated with a relationship line are cardinalities, which define the numeric relationships between occurrences of the objects

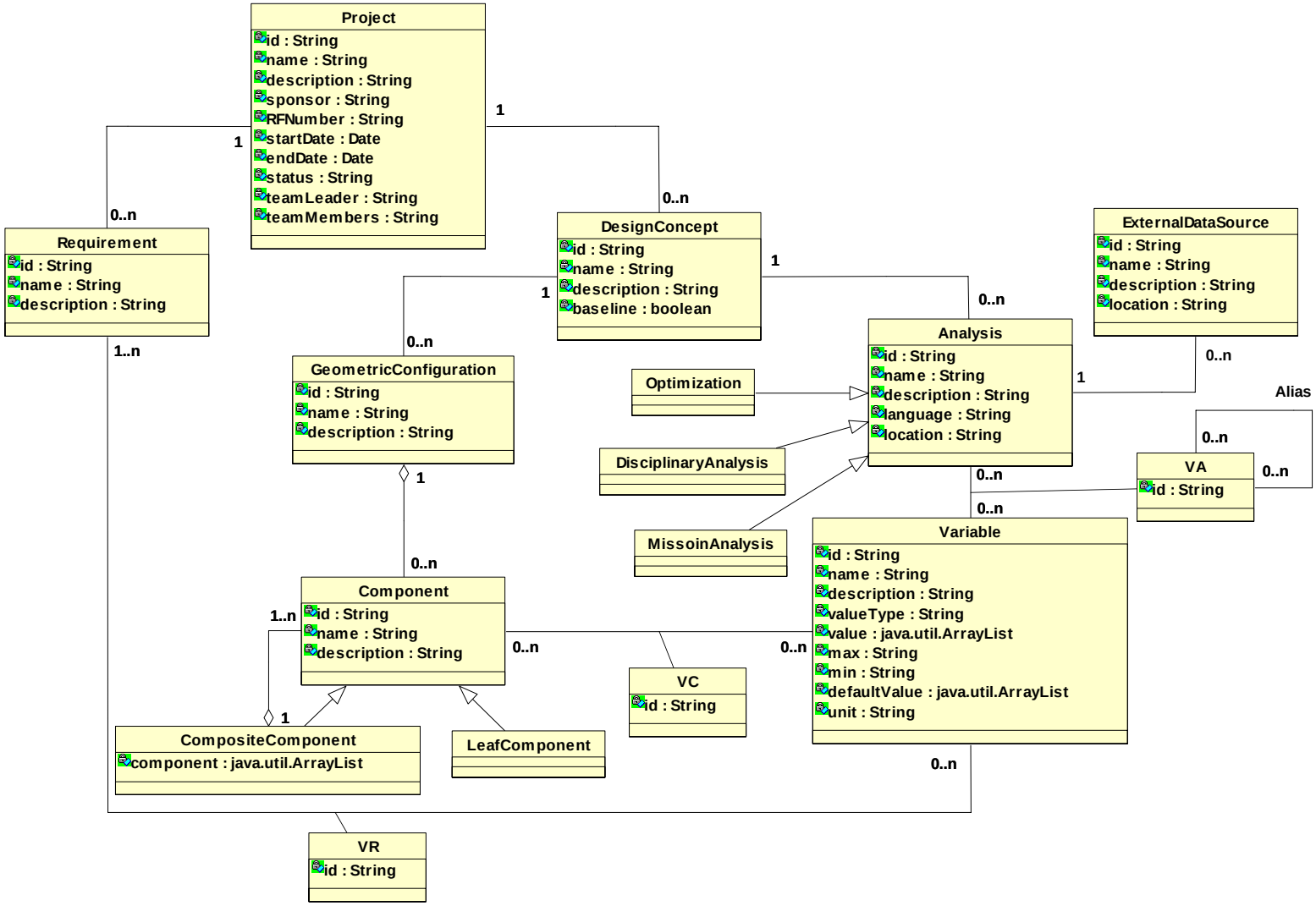

Figure 32. An Object-Oriented Data Model (with operations suppressed)

on either end of the relationship line. The relationships can be classified into five types: zero-to-one, one-to-one, zero-to-many, one-to-many, and many-to-many. For example, one design project can have zero or many design concepts, and one design concept usually belongs to one design project. The classes in this data model can be divided into two categories: data classes and relationship classes. Data classes are those classed used to model the logical data entities, such as a project, a design concept, or a variable. These classes include the "Project" class, the "Requirement" class, the "DesignConcept" class, the "Analysis" class, the "ExternalDataSource" class, the "GeometryConfiguaration" class, the "Component" class, and the "Variable" class. The many-to-many relationship of these data entities is modeled using relationship classes. These classes include the "VA" class, the "VC" class, and the "VR" class. They are used to model the many-to-many relationships among the classes of "Variable", "Analysis", "Component", and "Requirement. Each of these classes and it relationships with other classes in the model will be described in detail next.

- **"Project" objects:**

  This is a data class. As shown in Figure 33, a design is initiated as a "Project". It is the root element of the whole hierarchical data structure. Besides "id", attributes such as project name, project description, the sponsor of the project, the starting and finishing dates, etc. will be used to describe the project. For one project, which is an object of the "Project" class, there might be zero or many design concepts, which are instances of the "DesignConcept" class. However, one object of "DesignConcept" belongs to only one project. One "Project" object has

a set of requirements. Each requirement is treated as an object. And each requirement belongs to only one project. That is to say that the relationship between “Project’ and “Requirement” is one-to-many.

Operations shown here are for building the relationships with the classes of “Requirement” and “DesignConcept”. Accordingly, a set of “Requirement” objects and a set of “DesignConcept” objects need to be claimed as attributes for the “Project” class. Due to the limit of space, these obvious attributes are not shown in the above diagram for clarity. Please see appendix for example implementation of these classes.

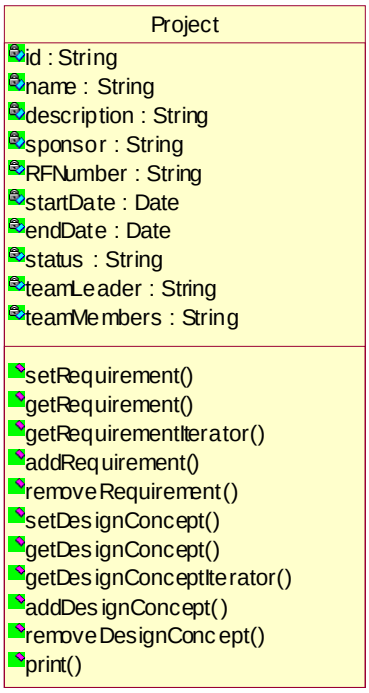

Figure 33. The "Project" Class

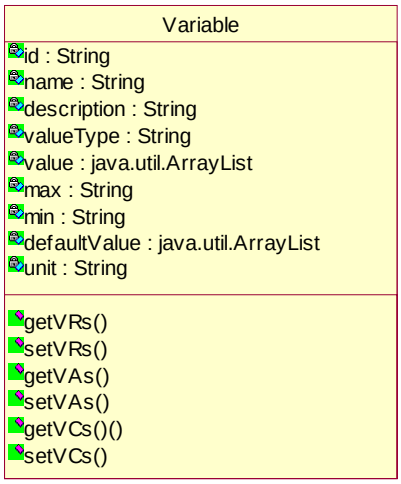

Figure 34. The "Variable" Class

- **“Variable” objects:**

  The “Variable” objects, which are modeled as data classes, are the leaf objects of the whole hierarchical architecture and cannot be decomposed further. The “Variable” class can be used to model both quantitative variables and qualitative variables. “Variable” objects are related to “Component”, “Requirement”, and “Analysis” objects through many-to-many relationships. Those operations shown in Figure 34, for example, setVAs(), setVCs(), and setVRs() are used to build the relationship with classes of “Analysis”, “Component”, and “Requirement”. “VA”, “VC”, and “VR” are termed as relationship classes since these classes are specifically created for modeling the relationships among “Variable”, “Analysis”, “Requirement”, and “Component”. The attributes used to describe a “Variable” object includes “id”, “name”, “description”, “valueType”, “value”, “unit”, etc. For a quantitative variable, the “valueType” is represented as a string, and “Value” is

modeled as “ArrayList” due to the fact that there are single-value variables and there are also multiple-value variables. This way, the value of a variable can be treated as the type specified by the string of “valueType”.

- **“Requirement” Objects:**

  Here, it is assumed that requirements can be described using either quantitative or qualitative variables. Therefore, “Requirement” objects shown in Figure 35 are modeled to have the many-to-many relationship with the “Variable’ objects. To explicitly represent this relationship, a new class “VR” is adopted in order to separate knowledge from operation [104].

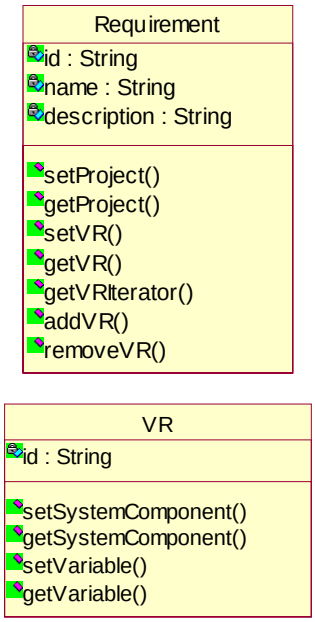

Figure 35. The "Requirement" Class and the “VR” Class

- **“DesignConcept” objects:**

  An object of “DesignConcept” as shown in Figure 36 can be described using attributes “id” (i.e., ID), “name”, and a boolean value, “baseline”, indicating if the design concept is the baseline concept or not. A design concept is related to both analyses, which are represented as “Analysis” objects, and certain geometry configurations, which are represented as objects of “GeometryConfiguration”. The relationship of a “DesignConcept” object with objects of “GeormetryConfiguration” is one-to-many, as is relationship with “Analysis” objects. Clarification should be made here. The synthesis and analysis of design concept involves many analysis tools. Indeed one analysis tool can be used for several design concepts. This is true. However, “Analysis” objects do not refer to a specific disciplinary analysis tool, but the analysis itself. A different analysis tool can be used for the same analysis. As will be described later, the “Analysis”

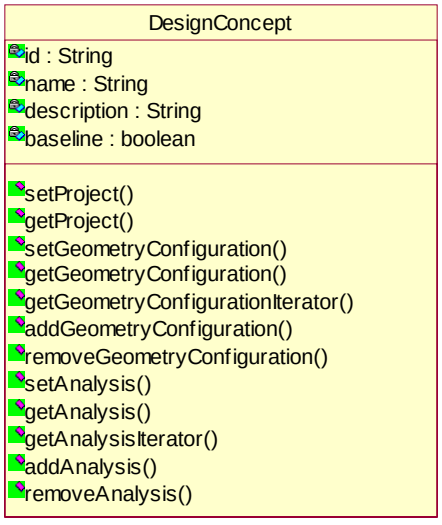

Figure 36. The "DesignConcept" Class

objects are also characterized by variables. For different design concepts, the variables involved are different even for the same disciplinary analysis using the same analysis tool.

Modeling the relationship between design concept and geometry configuration and the relationship between design concept and analysis this way enables a flexible decomposition of the information for a design concept. Sometimes, we prefer to decompose an aircraft based upon physical components. For example, an aircraft can be decomposed into fuselage, wings, tails, engines, etc. At other times, it is more convenient to decompose the analysis based upon disciplinary analysis involved. We can also decompose an aircraft design concept by combining both of these two approaches.

- **"Analysis" and "ExternalDataSource" objects:**

  An "Analysis" object is described using attributes of "id" (i.e. ID), "name", "description", etc. Just as an analysis involves many variables and one variable is shared by several analyses, "Analysis" objects are modeled to have the many-to-many relationships with the "Variable" objects. Because of the many-to-many relationship, we need an extra class "VA" to represent the relationship, which is the same as the relationship between "Requirement" objects and "Variable" objects. Besides this, it also happens that one variable can have different names in different analyses. The recursive relationship called "Alias" is used to model this situation. Also, analyses sometimes use external data sources, which are modeled as the "ExternalDataSource" class.

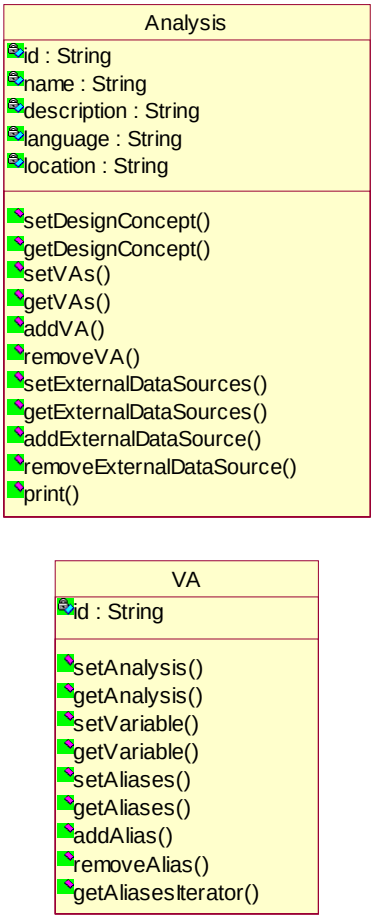

Figure 37. The "Analysis" Class and the "VA" Class

- **"GeometryConfiguration" objects:**

  The data class of "GeometryConfiguration" shown in Figure 38 is a superclass. It can be decomposed into different "Component" objects. For example, an aircraft geometry configuration can be decomposed into wings, fuselage, tails, canards, engines, etc. Each of these components is treated as an object, which is an instance of its subclass – "Component".

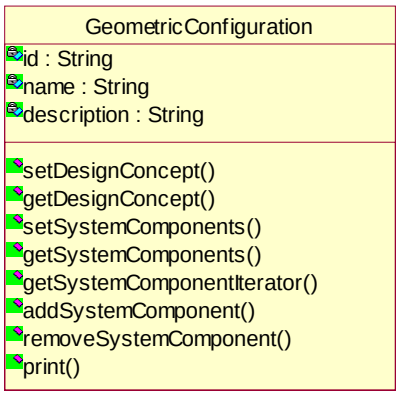

Figure 38. The "GeometryConfiguration" Class

- **"Component", "CompositeComponent", and "LeafComponent" objects:**

  The object model of component is a bit complicated to understand compared to other classes. Thus it needs further explanation. A component can be decomposed into subcomponents, for example, engine as a component can be decomposed into inlet, compressor, combustor, turbine, nozzle, etc., and in turn the subcomponent can be decomposed further; For example, the turbine as a component can be decomposed into subcomponents such as stator, rotator, etc. This relationship is modeled by decomposing a "Component" class into classes of "CompositeComponent" and "LeafComponent". At the same time, both of "CompositeComponent" class and "LeafComponent" class are subclasses of the "Component" class and inherent the attributes and operations of the "Component" class. A "LeafComponent" is atomic and cannot be decomposed further. However, "CompositeComponent" can. As illustrated in Figure 40, again, a "CompositeComponent" object consists of "CompsiteComponent" objects and "LeafComponent" objects. It is a recursive relationship.

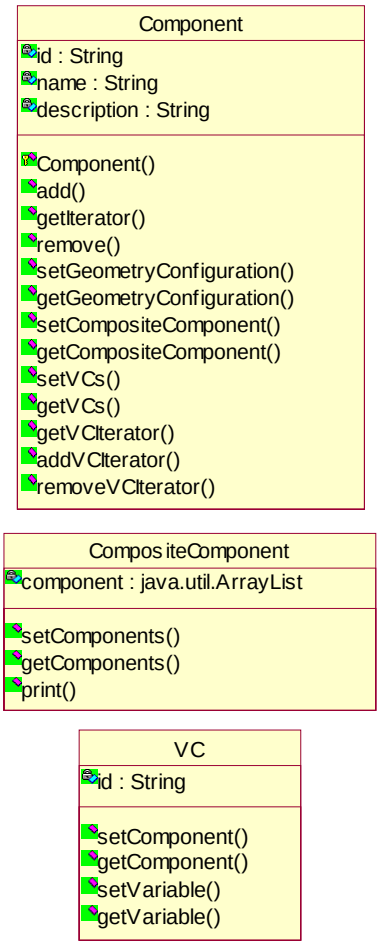

Figure 39. The "Component" Class, the "CompositeComponent" Class, and the "LeafComponent" Class

Beside its own attributes, such as “id” (i.e., identifier) and “name”, each “Component” object can be described using a set of “Variables” objects. Similar to the relationship between “Analysis” objects and “Variable” objects, their relationship is many-to-many. A class called “VR” is used to model this relationship.

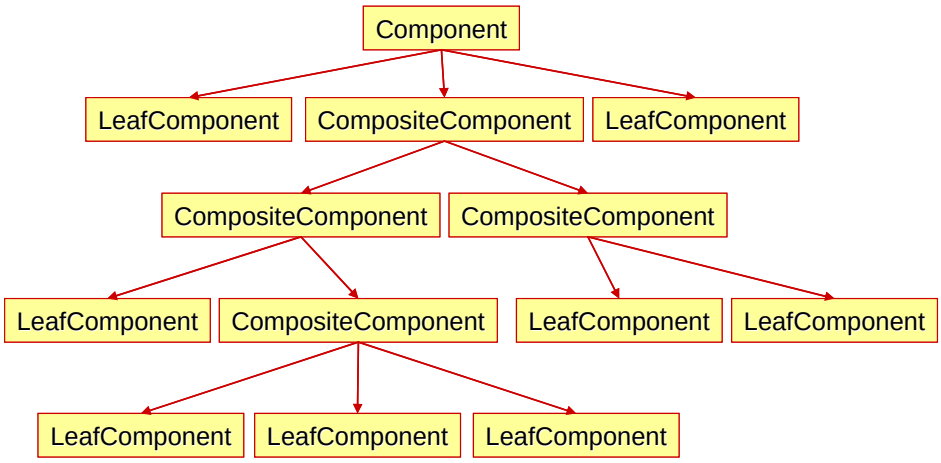

Figure 40. An Example of the Composite Structure of a Component

It has been mentioned previously that aircraft conceptual design data is characterized by a dynamic data structure. For different design concepts, the data structures are different, while for the same design concept; the data structure evolves with time. However, using current technologies, a database can to be built based upon a fixed data model or schema, which is the abstraction of the data structure. As discussed in the literature review section of Section 1.3.6, this is still an open problem and needs to be solved. During the course of this research, it been recognized that in order to overcome this barrier we need to work on the data model instead of waiting for the availability of enabling information technologies, that is to say we should build a data model that is robust enough for the evolution of conceptual design information and can be used for different design concepts. The building of such a data model requires a difficult tradeoff between the need for generality and the need to capture more detailed the information, for

the more detailed the information captured in a data model is, the more precise the model is able to describe a design.

The data model shown in the Figure 32 is a "well-balanced" data model. The abstraction of classes and the relationships among them are general enough for different design concepts, while the specific details of design data are included. For example, when designing a conventional aircraft with a major structure consisting of a fuselage, a wing, a vertical tail, and a horizontal tail, these components can be modeled using "Component" objects with "name" attributes of "fuselage", "wing", "vertical tail", and "horizontal tail" correspondingly. The "Variable" objects associated with these objects can capture other characteristics or specifications for these components. This is in contrast to the traditional way of modeling aircraft design data using components such as a "Fuselage" class, a "Wing" class, a "VerticalTail" class, etc, and using attributes of these classes to capture other information. Another example considers the design of an unconventional concept vehicle, such as a blend-wing-body (BWB) aircraft consisting of four main components: the center body, inner wing, outer wing, and winglet as shown in Figure 41. The data model proposed works well for this concept. Depending on how users name these components, they can be modeled as "Component" objects with the name attributes of "centerBody", "innerWing", "outerWing", and "winglet". And, "Variable" objects associated with these "Component" objects can capture other information such as the geometry specifications.

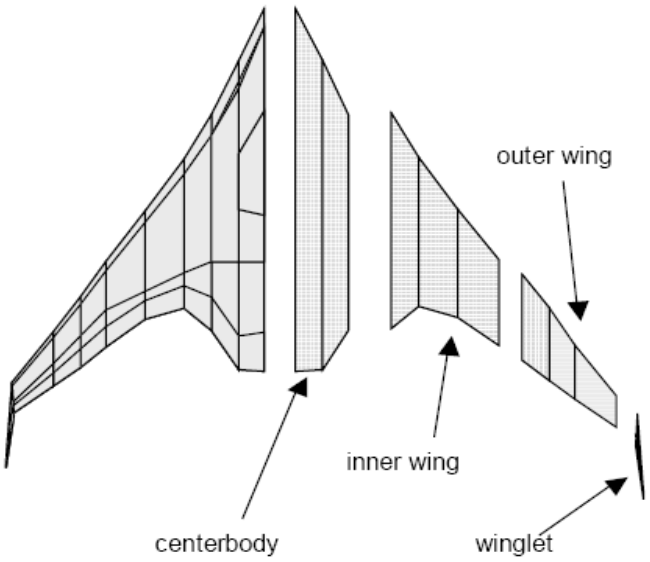

Figure 41. Modular Structural Breakdown of BWB [105]

One of the major differences between the proposed data model and how most researchers model aircraft conceptual design data is the separation of the "Variable" class. Previously, people model these variables within the component classes as attributes, for example the attributes of a "Component" class or an "Analysis" class. Separating these variables and modeling each of them as an object have the following benefits:

- It enables us to capture specific detailed information of the conceptual design data without sacrificing the generality of the data model.
- It makes data sharing and version control easier. For example, "wingspan" is a "Variable" object associated with a "Component" named "wing"; at the same time, this object can also be associated with an "Analysis" object called

"aerodynamicAnalysis". When the value of wingspan is changed, both the "wing" and the "aerodynamicAnalysis" are ensured to be updated simultaneously. They do not have to be managed separately.

- Data redundancy is reduced at the same time. The above example also shows that data redundancy can be reduced using this data model.
- It also makes it easier to extend the data model. As shown in Figure 42, by simply associating a "Variable" object with a "ProbabiltyDistribution" object, this model can be easily extended for probabilistic aircraft conceptual design and analysis.

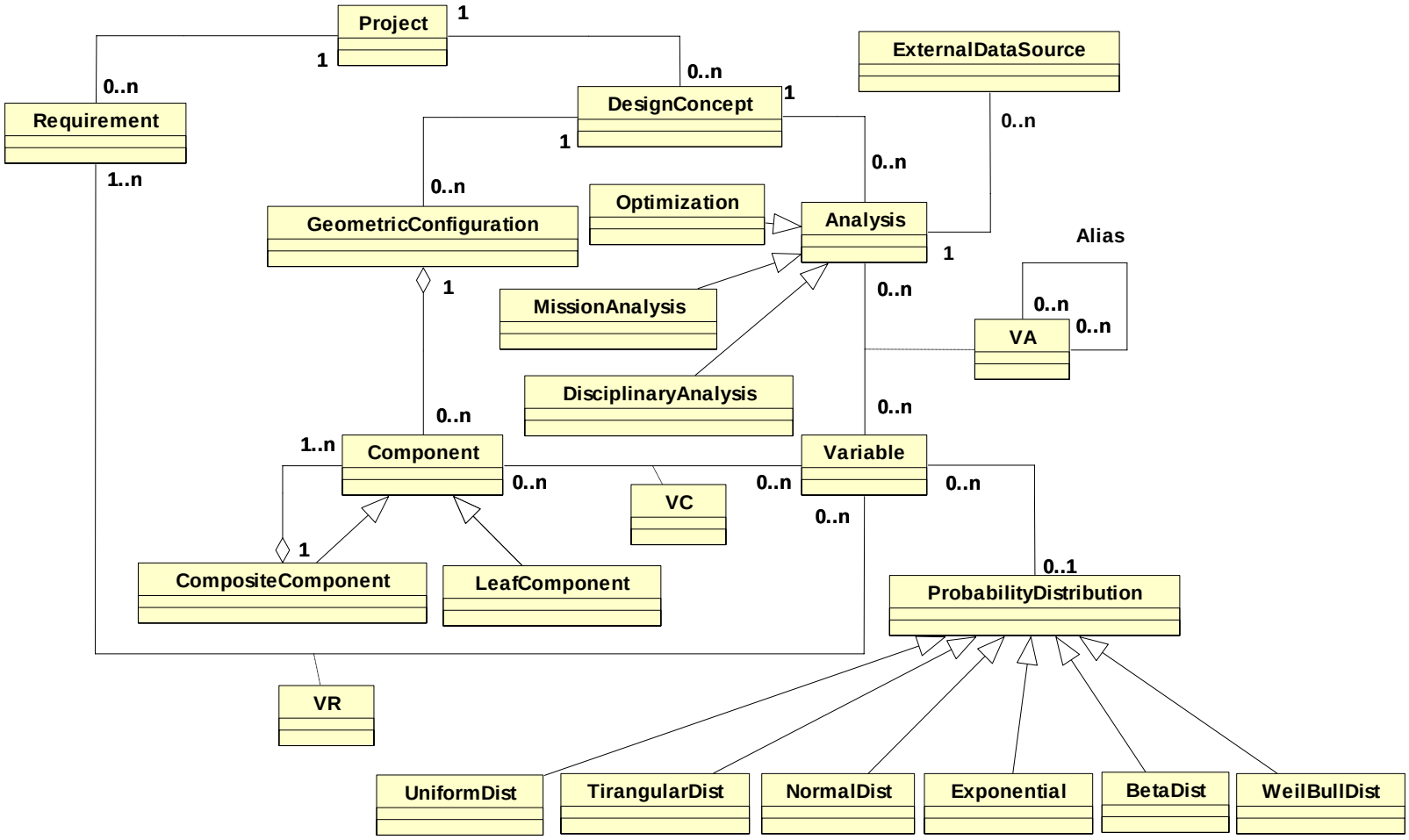

Figure 42. An Example to Extend the Data Model (with attributes and operations suppressed)

CHAPTER 6
IMPLEMENTATION

Starting with a high-level use case, this chapter presents the implementation of the proposed solutions to the research questions. In the object-oriented distributed framework of the NextADE, disciplinary analyses objects can be created by wrapping disciplinary analyses programs in existing legacy programs. It is the mission analysis component and the data management system that must be completely formulated. In this chapter, the implementation of the proposed object-oriented mission analysis approach will be presented first. Then based upon the proposed project-based object-oriented data model, a prototype data management system is developed.

6.1 Use Cases

Before we start to design the NextADE, this question is also asked in addition to the architecture and data management issues: what are the behavioral requirements for such a design environment? Obviously, besides trying to achieve those desired traits described in Table 5 and providing the functionality that designers need to accomplish those design and analysis tasks depicted in Figure 10 and Figure 11, the environment should also provide other general functions, such as allowing designers to create a new project and save it for later use etc. If the design environment is supported by a database, users

should be able to maintain the database. A UML use case diagram showed in Figure 43 models these behavioral requirements at a very high level.

This diagram describes the system's typical behavior as the system responds to a request from one of its stakeholders, i.e. the actors. The stick figures shown in the diagram are called actors. An actor is a role that a user plays with respect to the environment. The meaning of "user" is broader than what we are used to. An actor can be a person or an external system that plays a role in one or more interaction with the design environments. Here, a designer is an actor; an analysis tool treated as an external system that interacts with the design environment is also an actor. The oval shapes represent other use cases. An "include" relationship between two use cases means that a use case includes the functionality described in the other use case as parts of its process flow.

In the design environment of the NextADE, as shown in the diagram, a designer, who is an actor, can initiate three basic core use cases including "Create Project", "Do Analysis", and "Maintain Database". Practically, we manage the design of an aircraft as a design project. Accordingly, in this design environment, the design is managed on a project base. The project management activities are modeled using the "Create Project" use case. This use case includes the functionalities described in use cases of "Create Requirements" and "Create Design Concept". In turn, "Create Design Concept" includes "Create Geometry Configuration" and "Create Analysis Profile". The use case of "Create Analysis Profile" defines the design and analysis process by specifying when, how and where to call which analysis programs, and what design information they need. The "Do Analysis" use case captures the contract between the environment and designers about

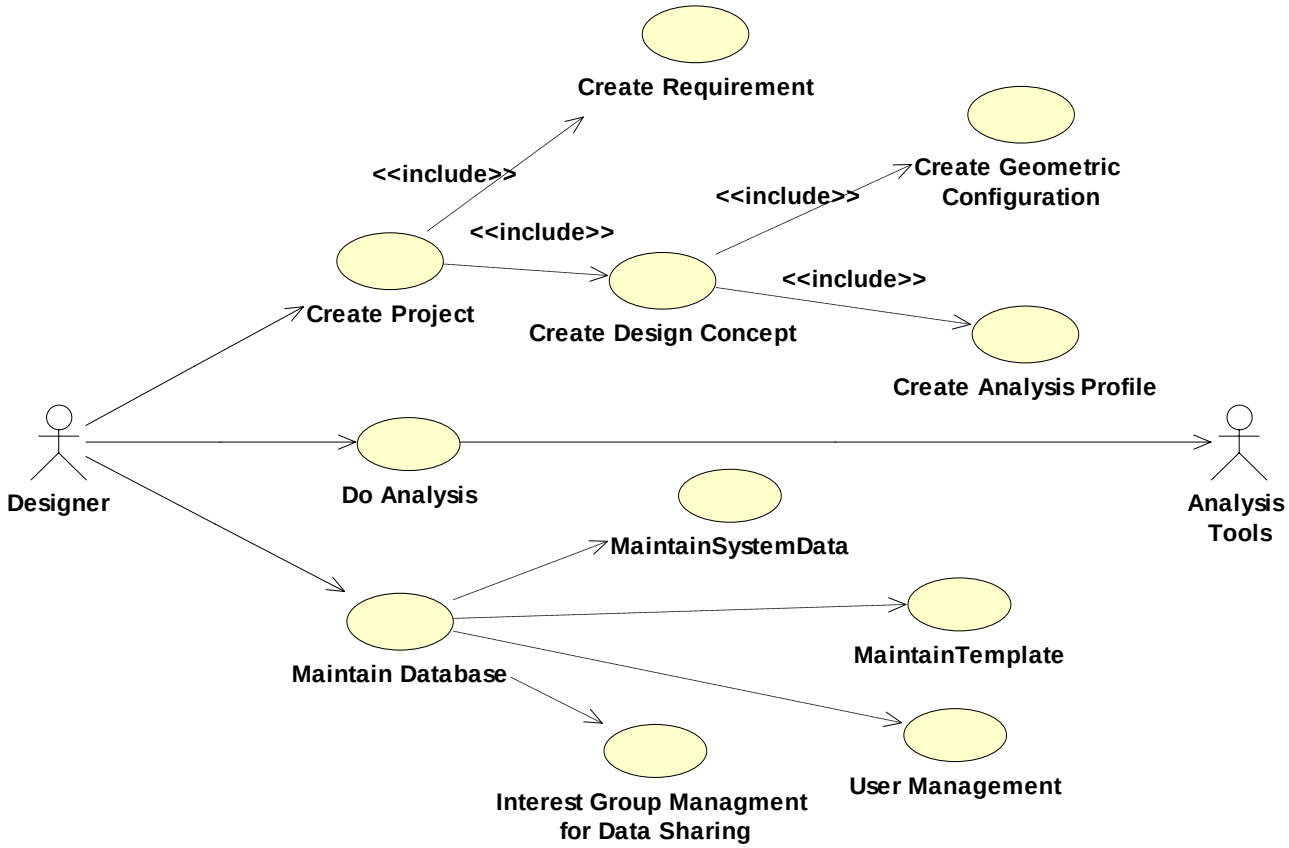

Figure 43. A High Level Use Case Diagram for the NextADE

how the system should behave under various conditions as it responds to designer's requests of invoking and executing analysis processes, which involves various analyses tools.

The "Create Project" use case is described using an UML activity diagram shown in Figure 44. In this use case, designers might input the description about the project first, for example, input the name, the starting date, the expected finishing date, a brief description of what the project is about, who sponsors it, etc. After that, the activities of use case "Create Requirement" and use case "Create Design Concept" will be invoked and executed. The information about the project will then be saved to the database.

The simple activity diagram shown in Figure 45 is the activity diagram for the use case "Create Requirement". When designers create a set of requirements for a design, the system will respond by reading the requirements either through the designer's input or by processing a file. Then the requirements will be saved to the database.

Since the NextADE is proposed to be supported by a data management system, it should provide functionality for designers to maintain the database. Use case "Create Project" is about recording information. Use case "Maintain Database" models the system behavior requirements that allow designers to maintain the database, i.e. managing the information. Its descendent use cases, "Maintain System Data", "Maintain Template", "User Management" and "Interest Group Management for Data Sharing", supply different implementations of the behavior of use case "Maintain Database". They also inherit the relationship between actor "Designer" and their ancestor, use case "Maintain Database". Shown in Figure 46 is an example activity diagram of "Maintain

Template". The environment should allow designers to conduct routine template maintenance activities, such as creating new templates, editing or deleting existing templates etc.

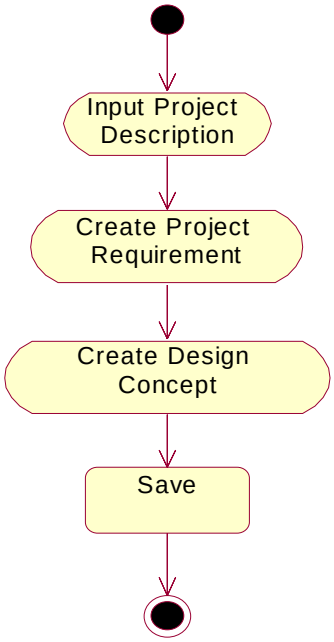

Figure 44. Activity Diagram of Use Case "Create Project"

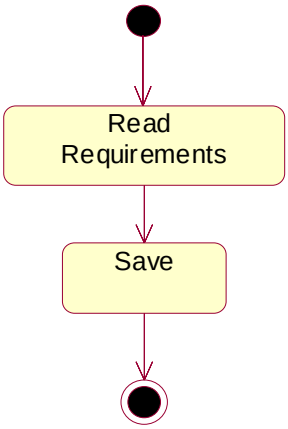

Figure 45. Activity Diagram of Use Case"Create Requirement"

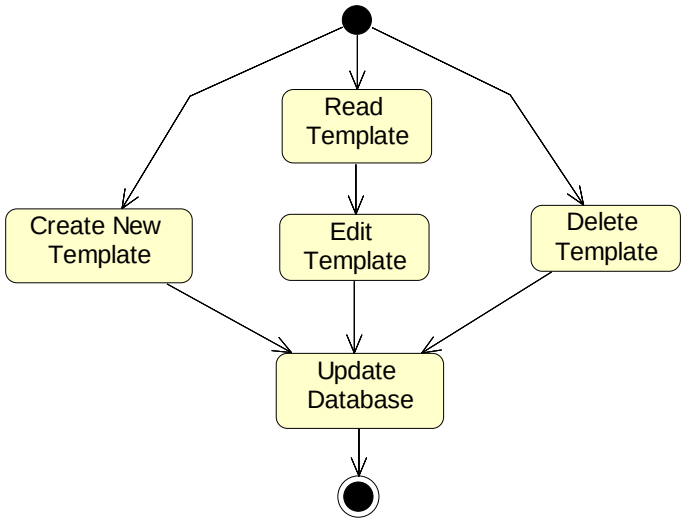

Figure 46. Activity Diagram of Use Case "Maintain Template"

These example use cases presented in this section only show the high level abstraction of the basic behavioral requirements of the NextADE. The purpose is to give a general idea of how we should start to "picture" the design environment so that the mission analysis component and the data management system can be implemented properly. These use cases should be extended and elaborated when we start to actually build a real design environment. The more detailed the behavioral requirements are that we capture in the use cases, the less trouble we will face in the later development stages.

6.2 The Mission Analysis Component

A class diagram showing the implementation of the mission analysis component is provided in Figure 47. A class diagram is one kind of UML (Unified Modeling Language) diagram as introduced in Section 3.2.3 of Chapter 3. In addition to the attributes and methods inside of each class, the static relationships of different classes are also shown. Due to the limitation of space, this diagram doesn't include every detail. For example, some trivial attributes, methods, and several classes are not shown. The wrappers for disciplinary analyses and the segment classes are written exclusively in Java, helping to ensure extensibility, compatibility, reusability, and platform independence. Please see Appendix A for the source code of the cruise segment class, as an example. Besides Java, other object-oriented programming languages, such as C++ and C#, can also be used. To demonstrate how different components interact with each other at run time, Figure 48 shows the sequence diagram for the required fuel weight calculation of the cruise segment. A sequence diagram is another type of UML diagram used to model the collaboration of objects at run time. From this diagram, it can be seen that the cruise object calls the *Atmosphere* object first, then the *Aero* object, and then the *Prop* object in order to calculate the fuel weight.

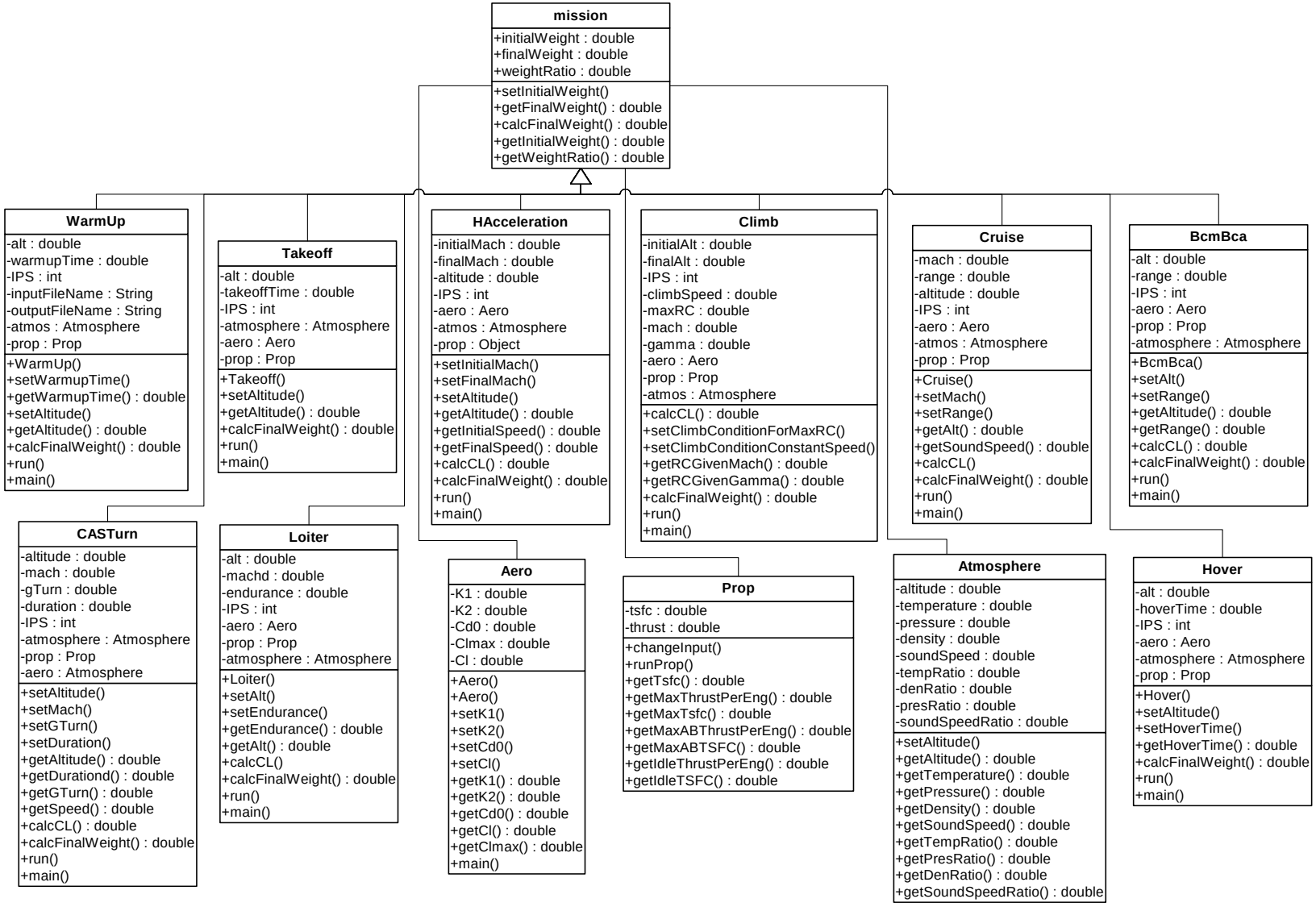

Figure 47. A Class Diagram of the Mission Analysis Component (partial)

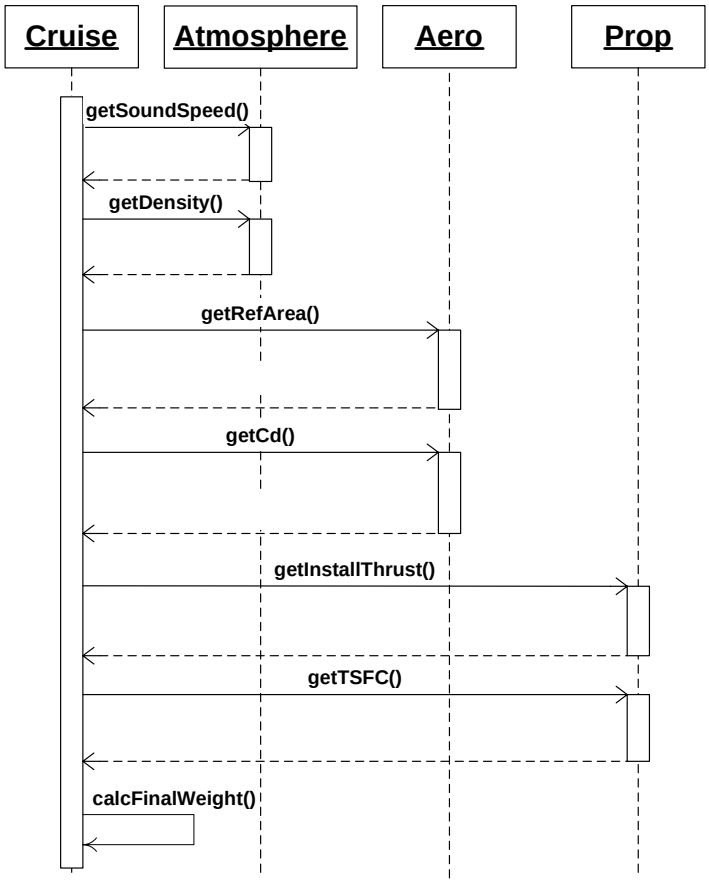

Figure 48. A Sequence Diagram for the Fuel Weight Calculation of Cruise Segment

6.3 A Prototype Data Management System

There are many ways to implement the object-oriented data model proposed in the previous section. For example, it could be implemented as an object-oriented data management system, a relational data management system, or an object relational data management system. However, considering the advantages and disadvantages of the enabling data management technologies, as has been discussed in Chapter 4, the

combination of object-oriented modeling, which is good to modeling engineering data, a relational database, which is a robust and mature technology for data management, and XML, which is versatile for data exchange, is employed as the implementation approach for this research.

By combining these information technologies, a prototype of the data management system has been developed. All the data object classes in the data model described previously are coded as Java classes. As implementation examples, the source codes for the "Variable" class, the "Component" class, and the "VC" class are given in Appendix E. Incorporating a data management system into the design environment is quite beneficial for designers to streamline the design process and manage design data. However, there is no free lunch. More flexibility for end users means more work for system architects and developers at the design and development stage. It is true that there is some overhead. For example, the environment is more complicated to build than one without the supporting data management system, and some efforts are also needed to maintain the database. Therefore, in order to reduce such overhead, the author suggests using a lightweight, easy to use relational database management system. In the market, there are many choices available. The open source relational database, MySQL [106], was chosen for the implementation here. In order to simplify the integration of the data management system, a user interface connecting to the database through XML data exchange is created. Other than these, the use of other computing technologies has also been explored, particularly, Hibernate [107], XDOCLET [108], CLI [109], and XERCES [110]. Please see appendices for brief descriptions of these technologies and the corresponding software tools.

6.3.1 Object Relational Mapping

In order to combine the object-oriented data model with a relational database properly, a strategy for transforming the object model to a relational model is required so that the data objects become persistent to the RDBMS. This is a problem that first appeared in the late 1980s [111]. It is called the object relational mapping problem and it refers to a programming technique that links SQL databases to object-oriented language concepts, creating (in effect) a "virtual object database". There are several solutions available to solve this problem including Java Data Object (JDO) by Sun Microsystems [112], TopLink by Oracle [113] , and Hibernate by JBoss Enterprise Middleware System [107]. Hibernate, which is open source, is used in the implementation. Hibernate enables users to develop persistent classes according to common Java idioms such as association, inheritance, composition, polymorphism, and the Java collections framework; and it also allows users to express queries in SQL [107, 114]. Using Hibernate, a persistent component needs to be added to the data management system architecture, which will be presented later in the architecture section. This component consists of classes that are responsible for data storage to and retrieval from the database.

In accordance with the object-oriented data model, the relational database is also a project-based database. A database called "oad" (stands for object-oriented aircraft conceptual design) is created in MySQL to store the design data. Shown in Figure 49 are the tables in the database corresponding to the object-oriented data model of Figure 32. Generally, each class is mapped to a table; the attributes of the each class are mapped to the attributes of the corresponding table/tables in the database; one-to-one relationships

are mapped through foreign keys which can be put in either table; one-to-many relationships are also mapped through foreign keys which have to be placed in the table on the many side; many-to-many relationships are mapped using extra tables. In the mapped tables, each row termed as a “tuple” in the relational database corresponds to an object of the class that the table represents.

```
mysql> show tables;
+----------------------------+
| Tables_in_oad              |
+----------------------------+
| analysis                   |
| composite_system_component |
| design_concept             |
| disciplinary_analysis      |
| external_data_source       |
| geometry_configuration     |
| leaf_system_component      |
| mission_analysis           |
| optimization               |
| project                    |
| requirement                |
| system_component           |
| va                         |
| va_aliases                 |
| variable                   |
| variable_values            |
| vc                         |
| vr                         |
+----------------------------+
18 rows in set (0.00 sec)
```

Figure 49. Tables in the Relational Database Corresponding to the Object-Oriented Data Model

For easy understanding, the tables in the database are named differently from their corresponding object classes. Following the naming convention of Java, the first letter in the name of each Java object class is capitalized; and if there is more than one word in the name, the first letter of each word is capitalized. However, for tables in the relational database, the names use all non-capitalized letters, and underscores are used to separate multiple words. The mapping of the classes in the object-oriented data model will be gone over in detail next.

- The mapping of the "Project" class, the "Requirement" class, the "DesignConcept" class, and their relationships:

  Shown in Figure 50 is the mapping. These three classes are mapped to three tables named "project", "requirement", and "design_concept" correspondingly. The attributes of these classes are mapped as the attributes of the corresponding tables. The attribute "id" (i.e. identifier), which is a UUID as described previously, is defined as the primary key of each of the tables. The one-to-many relationships of the "Project" class with "Requirement" and "DesignConcept" classes are mapped using foreign keys in the tables corresponding to the other two classes as shown by the connectors in the figure, i.e. the "project_id" attribute in table "requirement" is linked to the "id" attribute in the "project" table, and it is the same for the "design_concept" table.

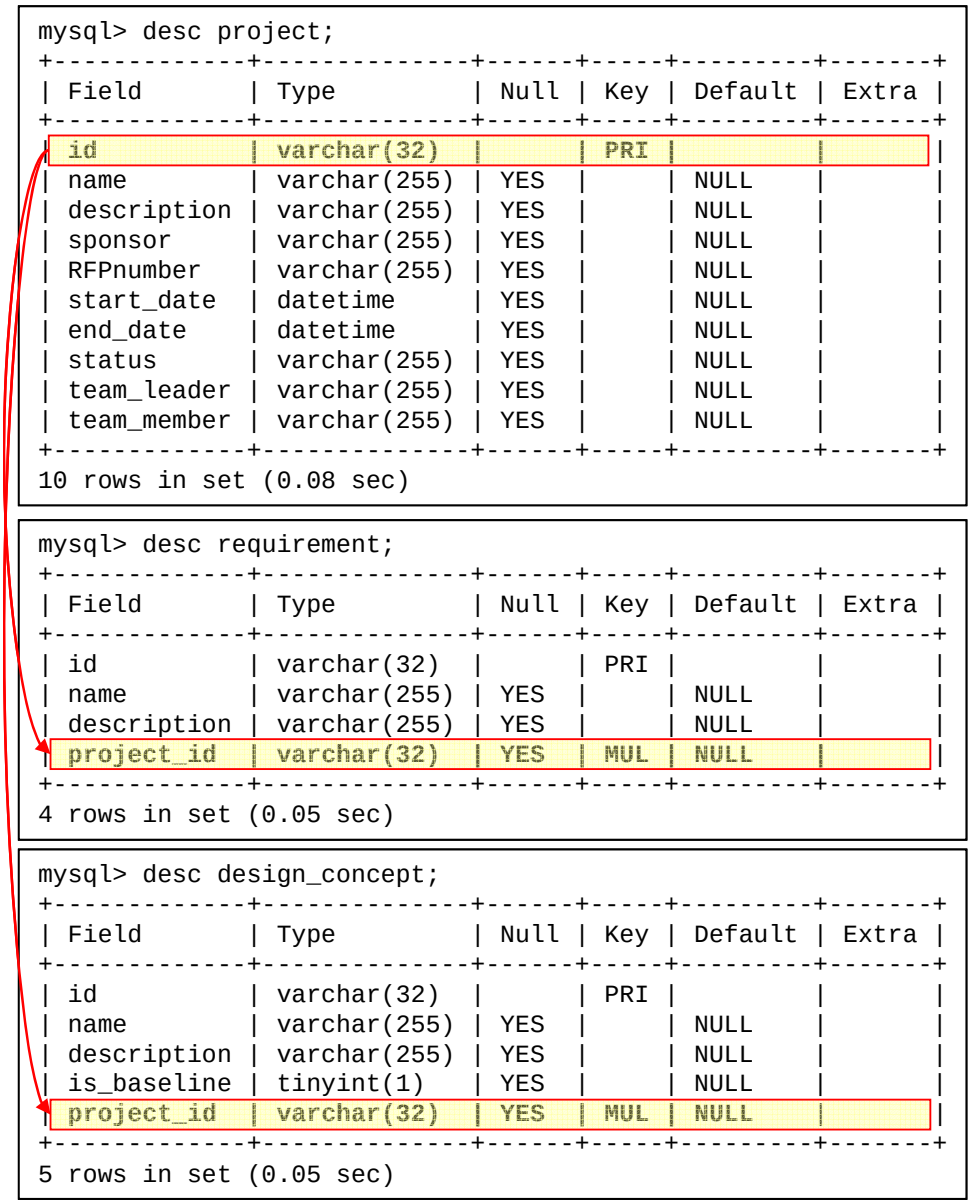

Figure 50. The Relational Mapping of the "Project"Class, the "Requirement" Class, and the "DesignConcept" Class

- The mapping of the “Variable” class:

  The “Variable” class is mapped to two tables named “variable” and “variable_value”, as shown in Figure 51. As has been discussed when describing the “Variable” class, a variable can be a single-valued variable or a multi-valued variable. In order to reduce redundancy, the “variable_values” table is used to store the value/values of the variables only. The primary key of the “variable_values” table is a compound one. It consists of an attribute called “variable_id”, which is linked to the primary key of the “variable” table, i.e. the “id” attribute, and an attributed named as “idx”, which is also used to trace how many values a variable has.

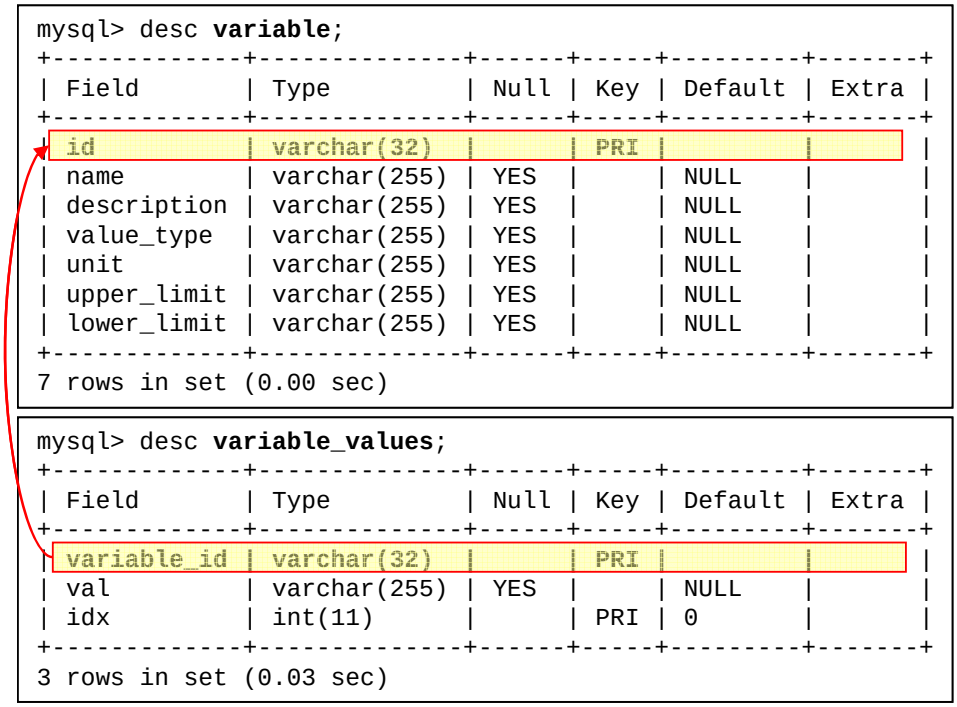

Figure 51. The Relational Mapping of the "Variable" Classes

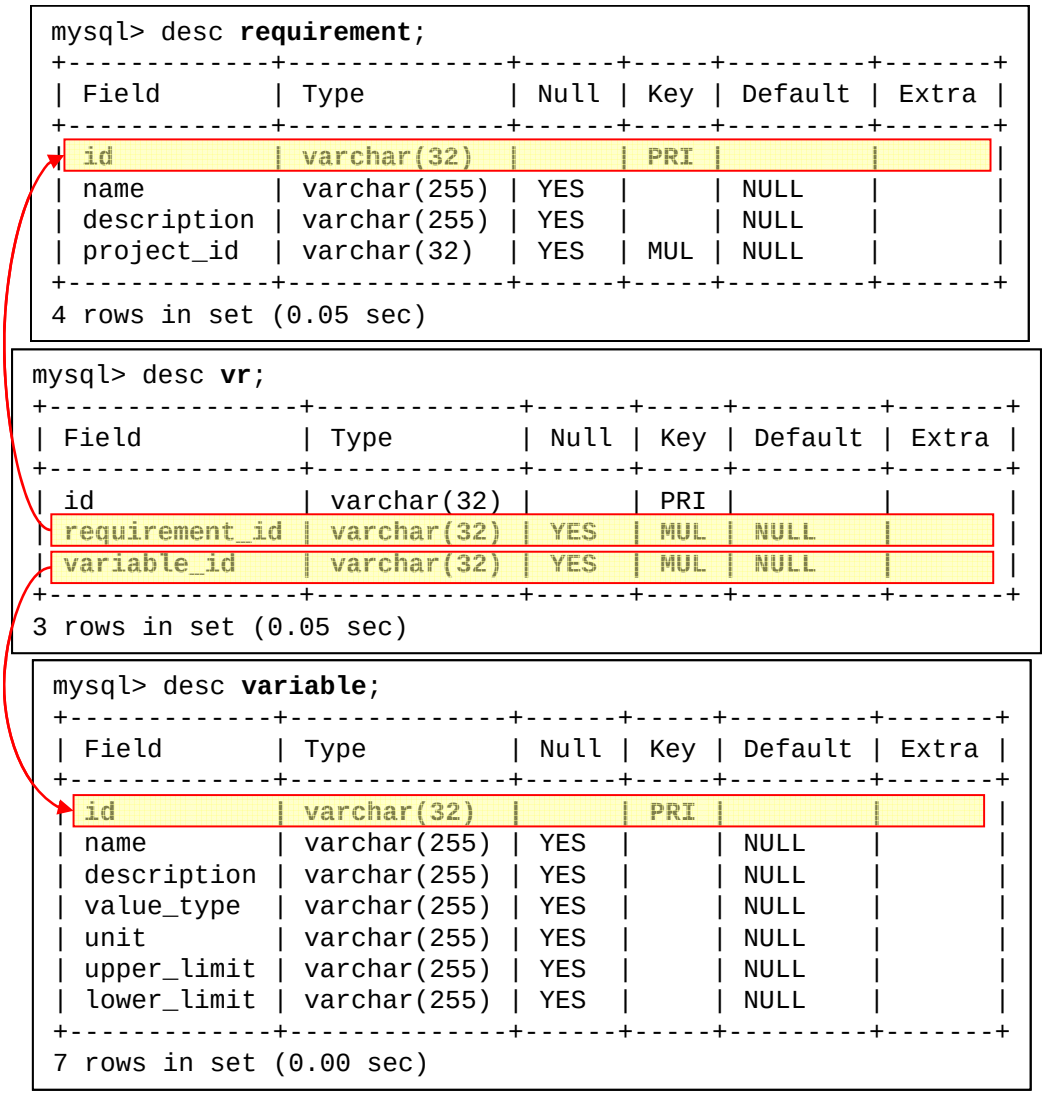

Figure 52. The Relational Mapping of the "Requirement" Class and the "Variable" Class

- The mapping of the relationship between the “Requirement” class and the “Variable” class:

  The relationship between these two classes is many-to-many. As shown in Figure 52, a table named “vr” is created for this relationship. In this table, besides the primary key, which is the attribute of “id”, two foreign keys are used to build the relationship. One is the “requirement_id” attribute, which is linked to the “requirement” table. The other one is the “variable_id” attribute, which is linked to the “variable” table.

- The mapping of the “GeometryConfiguration” class, the “SystemComponent” class, the “CompositeComponent” class, the “LeafComponent” class, and their relationships:

  Shown in Figure 53 is the mapping of these classes and their relationships. The relationship between the “GeometryConfiguration” class and the “SystemComponent” class is one-to-many. A geometry component may consist of several system components, and some of these components are actually the same. For example, an aircraft has two wings and the geometry configurations of the two wings are the same. To model this relationship, a foreign key named “system_component_id” is placed in table “system_component”. It is linked to table “geometry_configuration”. The attribute of “idx” is used to trace how many system components of the geometry configuration are the same, which works in the same way as the “idx” attribute in the “variable_value” table.

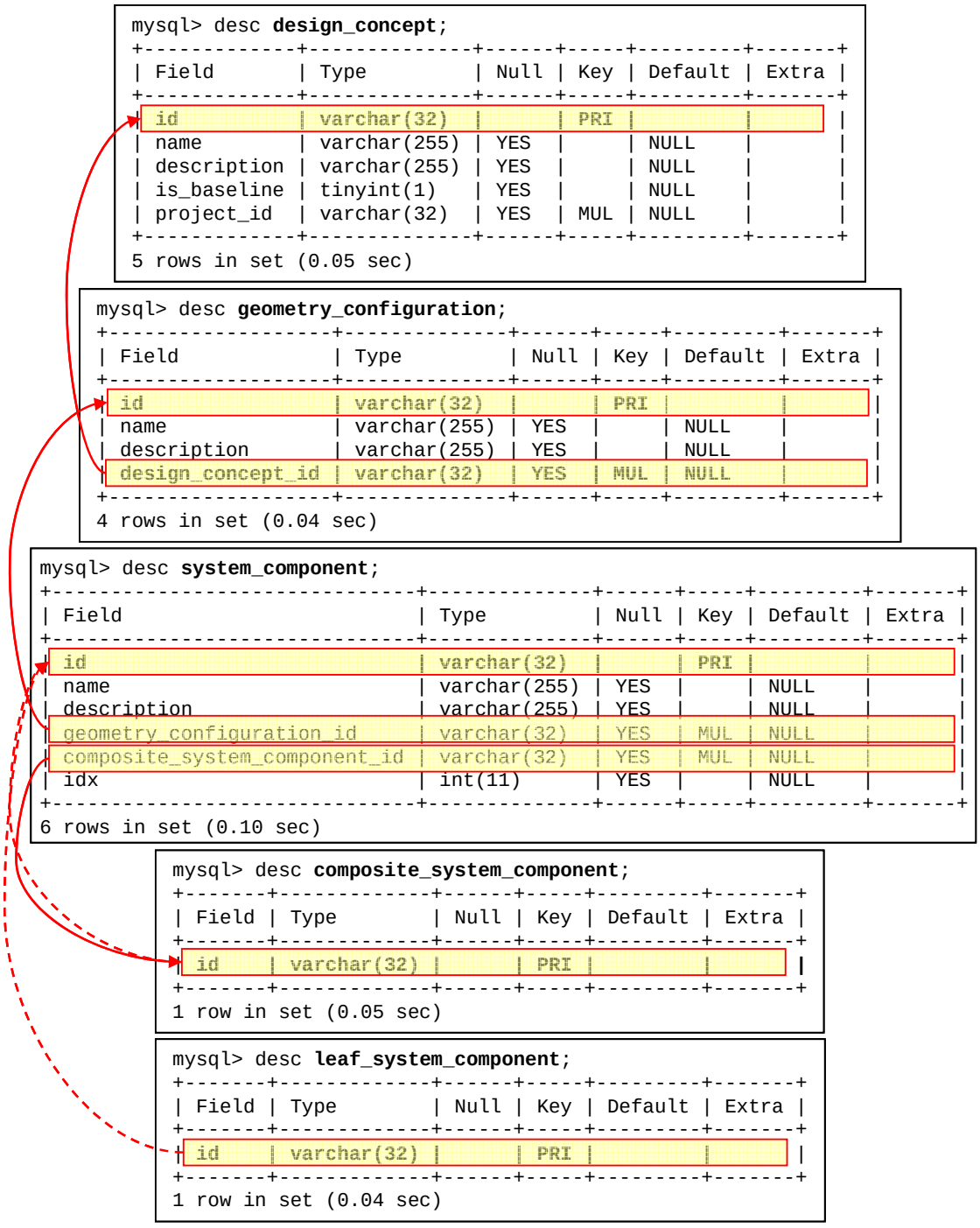

Figure 53. The Relational Mapping of the "GeometryConfiguration" Class, the "SystemComponent" Class, the "CompositeComponent" Class, and the "LeafComponent" Class

The "SytemComponent" class is the superclass of both the "CompsiteComponent" class and the "LeafComponent" class. This relationship is modeled by using joint "id" attributes in the corresponding tables, which means the values of "id" attributes for tuples in the table "composite_system_component" is a subset of the values of the "id" attributes for tuples in the "system_component" table. This is the same for the "leaf_sytem_component". Also, the recursive relationship between the "SystemComponent" class and the "CompositeComponent" class is mapped by linking the "composite_system_compoent" table with the "system_component" table through the attribute of "composite_system_component_id" in the "system_component" table.

- The mapping of the relationship between the "SystemComponent" class and the "Variable" class:

  This is a many-to-many relationship. A table named "vc" is created in order to model the relationship as shown in Figure 54. In the "vc" table, beside the primary key, there are two foreign keys: foreign key "system_component_id" is linked to table "system_component"; foreign key "variable_id" is linked to table "variable". This relationship is mapped in the same way as the relationship between the "Requirement" class and the "Variable" class.

.

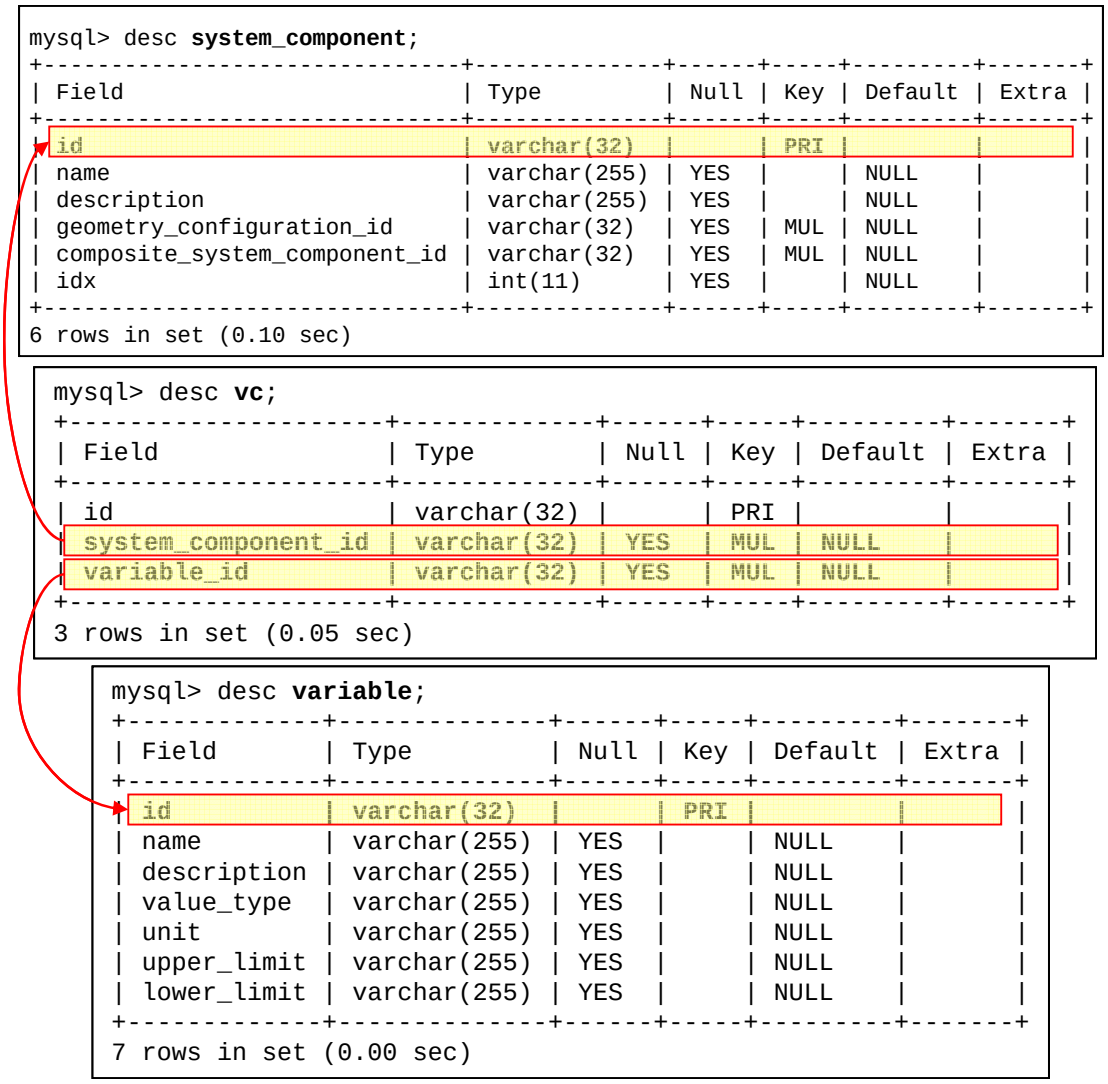

Figure 54. The Relational Mapping of the "SystemComponent" Class and the "Variable" Class

- The mapping of the “Analysis” class, the “Optimization” class, the “DisciplinaryAnalysis” class, the “MissionAnalysis” class, and their relationships:

  These classes and their relationships are mapped as shown in Figure 55. The classes are mapped to tables named “analysis”, “optimization”, “disciplinary_analysis”, and “mission_analysis” correspondingly. The “Optimization” class, the “DisciplinaryAnalysis” class, and the “MissionAnalysis” class are subclasses of the “Analysis” class. They inherent all the attributes of their parent class, the “Analysis” class. Their relationships are mapped using a joint primary key, which is the ‘id” attribute.

- The Mapping of the relationship between the “Analysis” class and the “Variable” class:

  This relationship is mapped as shown in Figure 56. Again, this is a many-to-many relationship. Table “va” is created to establish the relationship. In the table, except the two foreign keys which link to the two tables, two attributes are added to indicate if a variable is a control variable or just the analysis result. Since one variable may be used by more than one analysis with different names, table “va_alias” is created to store this relationship. Attributes “id1” and “id2” act as both the compound primary key and the foreign keys that refer to the “variable” table.

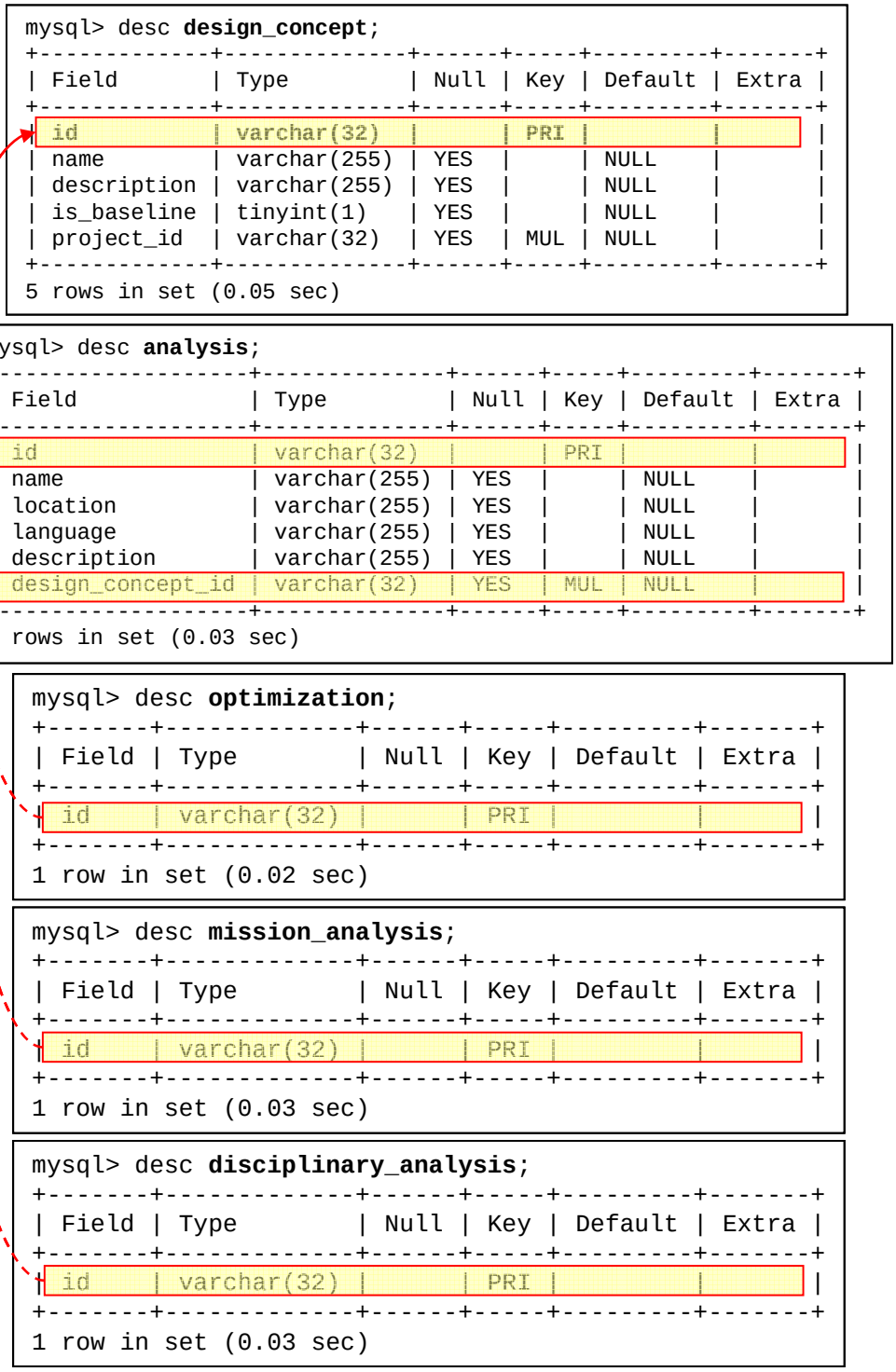

Figure 55. The Mapping of the "Analysis" Class and Its Subclasses

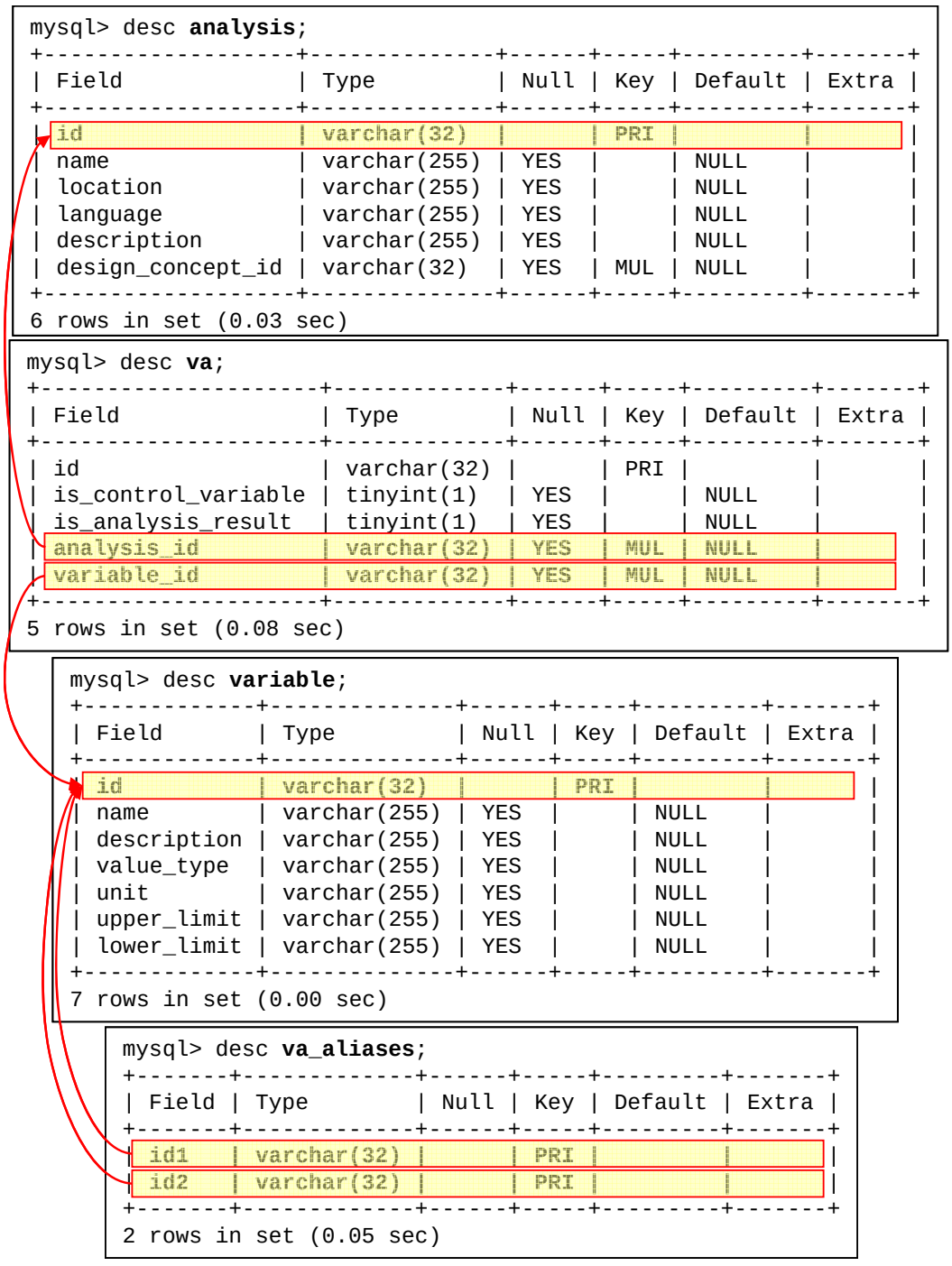

Figure 56. The Relational Mapping of the "Analysis" Class and the "Variable" Class

- The mapping of the “ExternalDataSource” class and its relationship with the “Analysis” class:

  The “ExternalDataSource” class and its relationship with the “Analysis” class are mapped as shown in Figure 57. Since the relationship is modeled as one-to-many, only a foreign key, which is the “analysis_id” attribute, needs to be created on the many side, which is the table of “external_data_source”.

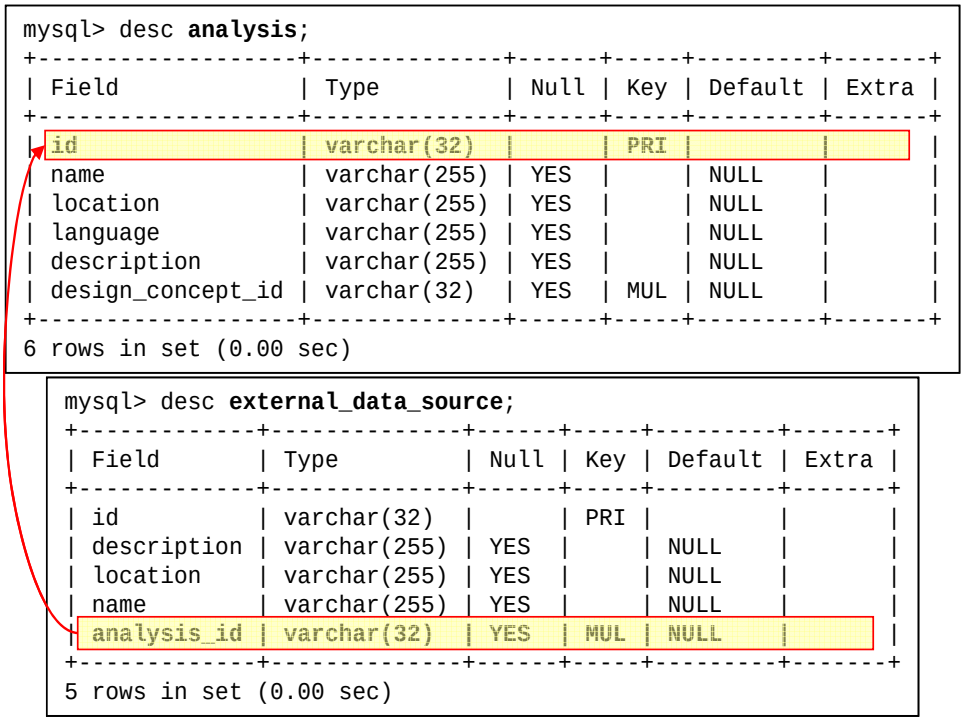

Figure 57. The Relational Mapping of the "ExternalDataSource" Class and the "Analysis" Class

6.3.2 The Application Architecture

The implemented prototype data management system has an application architecture as shown in Figure 58. It is a layered architecture. According to Trowbridge et al [115], a layered architecture is a logical structuring mechanism for the elements that make up the software solution. The software application elements of the prototype data management system are organized into three layers: the presentation layer, the application layer, and the data layer. Shown in Figure 59 is a UML package diagram of the implemented prototype of the data management system. As previously introduced, a package diagram reflects the organization of the packages and their elements. It is actually another form of the class diagram, which is at a higher level. The dashed arrows represent dependency relationships among the packages. Dependency relationships exist when changes to the definition of one package require changes to the definition of other packages. For example, changes to the definition of either the "pobj" package or the "service" package may cause changes to the definition of the "client" package. Therefore, the "client" package depends upon the "pobj" package and the "service" package. As shown in the figure, all the Java classes of the implementation are organized into six packages based upon their functionalities. Each package is dedicated to a specific layer in the application architecture. Following is a description about the functionality of each layer and its corresponding packages:

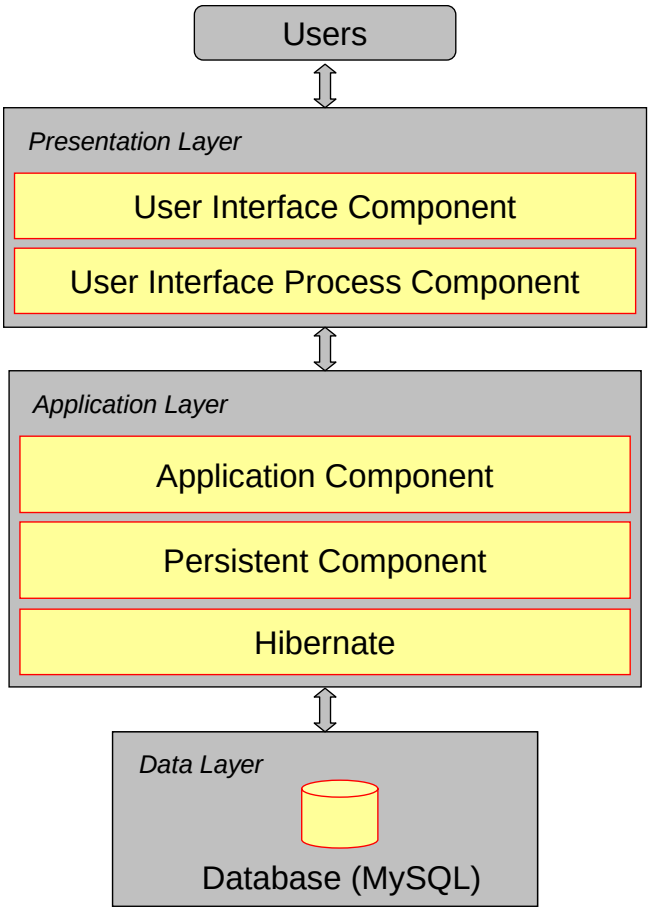

Figure 58. The Application Architecture for the NextADE Prototype

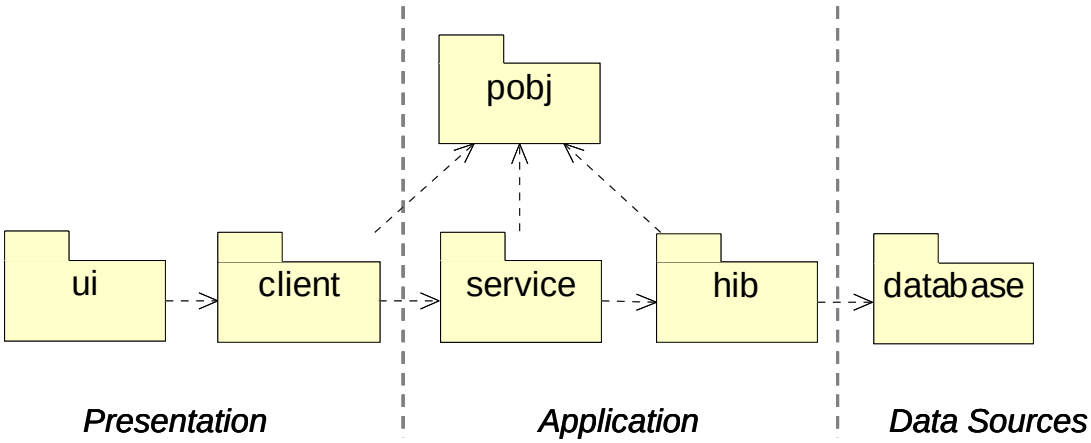

Figure 59. A Package Diagram of the Data Management System

- The presentation layer provides the user interface of the system. Package "ui" (short for user interface) together with package "client" provides the functionality of this layer. Included in the package of "ui" are the classes for handling user input from the command line. Package "client" then processes user requests captured by the "ui" package. For this prototype data management system, a GUI was not developed since its design and development is not the focus of the research. However, a simple command line user interface has been developed, and it will be presented in detail in the next section.
- The application layer implements the application functionality of the system. It is composed of the components of application classes, persistent classes, and Hibernate, which correspond to the packages of "service", "pobj", and "hib" in the package diagram correspondingly. The "service" package

includes the database control classes. It depends on both the "pobj" package and the "hib" package. Package "hib" is Hibernate, which is used for mapping the persistent data objects into a relational database. It relies on the critical "pobj" package. All the persistent data object classes shown in Figure 32 are organized into this package.

- The data layer is where the MySQL database resides and it also provides access to the database. In the implementation, the functionality of this layer is achieved through the package called "database".

6.3.3 The User Interface

The user interface of the data management system can be created in several different ways. Generally, if we want the data management system to be a web based system and support distributed activities across the Internet, we can develop a browser based interface; if we want the system to be a standalone application, a GUI can be created for it. For this prototype implementation, a command line user interface is created. Shown in Figure 60 is a snapshot of the interface. The interface is simple and easy to implement, and the functions can be executed in batch mode, which makes it easier to integrate the data management system to the design environment. The data operation functions provided by the prototype system meet the needs of the experiments conducted in the next chapter. In the future, besides developing a GUI, we should also keep such functionality in order to improve the interoperability of the system.

```
D:\OAD\jar>java -jar oad.jar
usage: java -jar oad.jar [options]
 -a,--analysis <analysis>      analysis name
 -c,--comp <component>         component name
 -d,--design <designconcept>   design concept name
 -e,--erase                    erase...
 -g,--geo <geometry>           geometry name
 -h,--help                     print this message
 -l,--list                     list...
 -p,--proj <project>           project name
 -r,--req <requirement>        requirement name
 -s,--save <file>              save xml file to database
```

Figure 60. The Command Line User Interface of the Data Management System

This interface enables users to query the database. It also provides some functionality to manage the database. Obviously, the database can be managed in MySQL itself. However, designers do not necessarily have to deal with MySQL directly and they do not need to learn MySQL. These functions provide an easy and simple interface for the supporting database of MySQL.

Starting from a list of projects, the data in the database can be queried based upon the hierarchical structure of the data. Listed in Table 7 are the basic command line commands that are available at this development stage and can be used to query and manage the database. More commands can be added to the system for future development. This implementation project is named "oad". The execution file, "oad.jar" is the compressed file that includes all class files that belong to different packages in the project. For the commands listed in the table, arguments shown in *italic* must be replaced by whatever they represent. They are actually the names of objects. The options in these commands cannot be omitted. They must be entered. The meaning of the options are displayed when using command "java –jar oad.jar", as shown in Figure 60, or command

"java –jar oad.jar –h". The execution results of the commands listed in the table will be presented in the Chapter 6 when the database is populated for case studies.

Table 7. Command Line Commands

Query Commands:

`Java -jar oad.jar –l`
List the names of the design projects that are stored in the database

`Java -jar oad.jar -l -p` *`projectName`*
List the names of all requirement lists and all design concept lists under the specified design project

`Java -jar oad.jar -l -p` *`projectName`* `-r` *`requirmentListName`*
List the requirements, which are represented in the form of variables, that belong to the specified requirement list of the specified design project

`java -jar oad.jar -l -p` *`projectName`* `-d` *`designConceptListName`*
List the names of all the geometry configuration lists and the analysis lists related to the specified design concept of the specified design project

`java -jar oad.jar -l -p` *`projectName`* `-d` *`designConceptName`* `-g` *`gometryConfigurationListName`*
List the names of all the system components consisting of the geometry configuration that belongs to the specified design concept of the specified design project

`java -jar oad.jar -l -p` *`projectName`* `-d` *`designConceptName`* `-g` *`geometryConfigurationListName`* `-c` *`systemComponentListName`*
List the names, the values, the units, etc. of all the variables of the specified system component consisting of the specified geometry configuration of the specified design concept of the specified design project

`java -jar oad.jar -l -p` *`projectName`* `-d` *`designConceptName`* `-a` *`analysisListName`*
List the names, the values, the units, etc. of all the variables related to the specified analysis of the specified design concept of the specified design project

Data Management Commands:

`java -jar oad.jar -l -s` *`xmlFileName`*
Store the data from an XML file into the database

`java -jar oad.jar -l -e` *`projectName`*
Delete the data stored for a design project

6.3.4 Data Exchange

As introduced earlier, XML provides an application independent way of sharing data. Unlike relations and objects, XML does not need a specialized serialization process to transfer the data and relationships. In order to make it easier to integrate the data management system with the design environment and populate the database, functionality for data exchange using XML files is added to the data management system.

Based upon the object oriented data model shown in Figure 32, a Document Type Definition [116] has been created to define the structure of the XML file with legal building blocks. With this DTD, different groups of users can agree to use a common DTD for data exchange. The system can also use it to verify if the data file is valid. A DTD file consists of five building blocks: 1) elements, which are the main building block; 2) tags, which are used to markup elements; 3) attributes, which provide additional information about the elements; 4) entities, which are variables used to define common text; 5) PCDATA, which refers to character data that will be parsed by a parser; and 6) CDATA, which refers to character data that will not be parsed by a parser.

Shown below is part of the XML DTD file for the purpose of explanation. The definition of some of the attributes is omitted. Please see Appendix G for the complete DTD file. This DTD is provided as an external file to the data management system.

```
================================================================
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com)
       by Zhijie Lu (Georgia Institute of Technology) -->
<!ELEMENT project (requirement | designconcept | variable)*>
<!ELEMENT requirement (vr)*>
<!ELEMENT designconcept (geoconfig | optimization | discipanalysis |
missionanalysis)*>
<!ELEMENT geoconfig (comsyscomp | leafsyscomp)*>
```

```
<!ELEMENT comsyscomp (comsyscomp | leafsyscomp)*>
<!ELEMENT leafsyscomp (vc)*>
<!ELEMENT optimization (analysis)>
<!ELEMENT discipanalysis (analysis)>
<!ELEMENT missionanalysis (analysis)>
<!ELEMENT analysis (extdatasrc | va)*>
<!ELEMENT variable EMPTY>
<!ELEMENT va EMPTY>
<!ELEMENT vc EMPTY>
<!ELEMENT vr EMPTY>
<!ELEMENT extdatasrc (#PCDATA)>

<!ATTLIST project
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST requirement
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>

......

=====================================================================
```

Besides the header part at the top of the DTD file shown above, this file can be mainly divided into two sections. One section is used to define the legal elements and their relationships. These elements and relationships are defined within each <ELEMENT … > tag. For example, the element named "project" corresponds to the "Project" class in the data model. The "project" element has three subelements, which are the "requirement" element, the "designConcept" element, and the "variable" element. Each of these subelements has multiple occurrences, which are represented by "*'. This models the relationship among the "Project' class, the "Requirement" class, the "DesignConcept" class, and the "Variable" class. The other section declares the attributes

of these elements. These attributes are defined using tag <!ATTLIST … >. For example, there are nine attributes defined for the "project" element. These attributes correspond to the attributes of the "Project" class shown in Figure 32; the identification attribute "id" (i.e. ID) must be specified; the other eight attributes are not required nor do they have default values.

The data structure defined in this DTD is a mapping of the data model shown in Figure 32 from the object-oriented format to the XML format. If we implemented this data management system using an XML database, this DTD is actually the data model of the database. As XML technologies, especially XML query language, become more mature, this should be one of the future research avenues.

6.4 The Distributed System Infrastructure

As shown in the mission analysis validation example, the NextADE can be integrated using commercial integration software packages, for example *ModelCenter*. However, in order to fully utilize domain expertise and have more aircraft conceptual design-specific capabilities, it is desirable to build the NextADE as a dedicated, standalone, self-contained design environment. For this purpose, following the application structure shown in Figure 58, a distribute system architecture is proposed for the NextADE, as shown in Figure 61. Since the realization of this system architecture is not the prime focus of the research reported in this dissertation, the proposed architecture described in this section only serves as a reference.

Depicted in Figure 61 is a three-tier client-server information system infrastructure. The client tier hosts the presentation layer components, and interacts with the users of the design environment. For Web applications, this tier consists of the user workstation, personal digital assistants (PDAs), and other devices that host a Web browser. For a rich client application, it consists of the computers and other devices. The software on this tier should provide a mechanism for presenting user interface and software infrastructure required to communicate with other tiers. The middle tier is the application tier. This tier hosts the design logic layer and the persistent layer, which includes those design and analysis components, and components that keep data persistent. For a Web application, it includes the Web servers as well. The hardware devices in this tier are computers of Web servers and application servers. The application server software installed on the application server should provide the execution context for the design logic; the Web server software should provide security and network connectivity. Correspondingly, the data layer is hosted in the data tier. In this tier, there are servers that host the database management system and other external computing resources that the design environment needs.

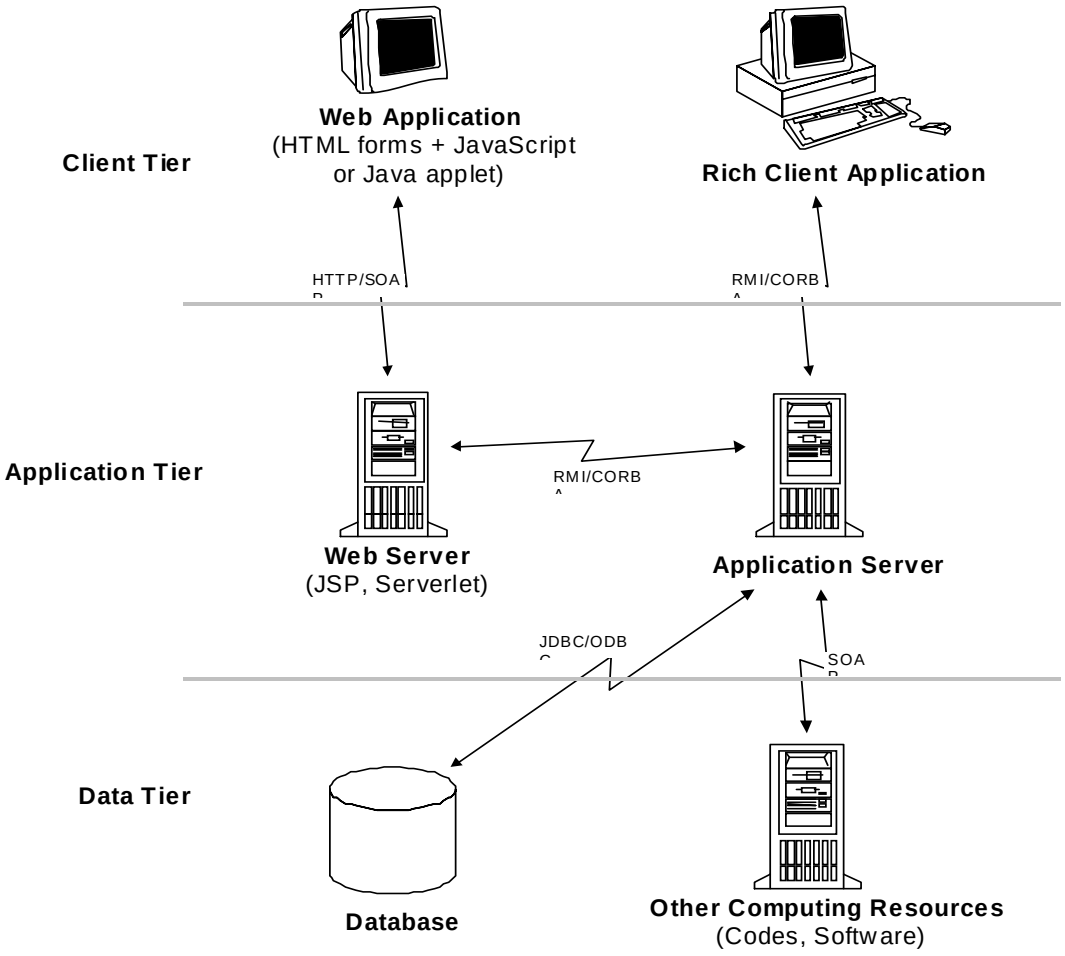

Figure 61. Multi-tier Distributed Information System Architecture

CHAPTER 7

EXPERIMENTS

In order to demonstrate the efficacy of the proposed object-oriented aircraft conceptual design framework and the data management system described in the previous chapter, two very different case studies are conducted in this chapter. One case study is based upon a simplified conceptual design problem of a notional conventional aircraft. Another case is a simplified conceptual design problem of an unconventional aircraft – a hybrid propulsion system powered small aircraft. As a separate building block, the data management system is integrated into the object-oriented design framework for both case studies. The database is populated with the design data. Design processes with the support of the database management system are then built. In order to reduce the effort of designing and developing a GUI and deploying the whole system architecture, at this stage, *ModelCenter* is used as the prototype integration tool for the experiments of this research. As described in Section 1.3.7, *ModelCenter is* a commercial software system which can be used to integrate analysis tools through wrappers, which process the input and output files of the analysis tools. It should be noted that other general-purpose integration software packages might do equally well for the same purpose.

7.1 Procedure of Integration

A general-purpose integration software package, *ModelCenter*, is used as the vehicle for the prototype integration of the proposed object-oriented distributed framework. Before the integrated design environments are presented for the two case study examples, a step-by-step procedure for the integration of the design environment using *ModelCenter* is provided first in this section.

Besides the data management system described in the previous chapter, the building blocks of the integrated design environment also include the disciplinary analysis component and the mission analysis component. The disciplinary analysis component includes analyses programs for optimization, aerodynamic analysis, atmospheric property analysis, engine cycle analysis, and geometry design. The mission analysis component consists of analyses programs for mission segments of warm-up, takeoff, climb, cruise, turn, loiter, hover, etc.

A prototype object-oriented design environment is constructed by integrating these building blocks using *ModelCenter*. Listed below are the steps, customized to *ModelCenter,* which can be followed in order to integrate the building blocks:

Step 1: Create a database in MySQL to store the design data, assuming that MySQL has been installed. A database named "*oad"* was created in MySQL for the two case studies, each of which corresponds to a project in "oad" per Figure 32.

Step 2: Identify and collect information/data that need to be stored into the database. The information includes such design data as the required

mission, the initial geometry configuration, performance requirements, other physics based attributes, etc.

Step 3: Choose analyses tools. Based upon the required fidelity, proper analysis programs for disciplinary analyses and mission analyses should be used. Once these analysis programs are chosen, besides gathering the information such as where these analysis programs are installed and how to execute them, we also need to identify the input/output information for each analysis program.

Step 4: Store data collected in Step 2 and Step 3 into the database created in Step one. Input these data into an XML file first. The XML file is used for data exchange with the data management system. It must conform to the XML DTD given in Appendix G. Then store the data in the XML file into the database created in Step 1 by running the command line command provided by the prototype data management system:

"*java –jar oad.jar –s fileName.xml*"

To make this task easier, a flexible computer-aided approach to input data into the XML file is to use an XML editor, such as *XMLSPY [117],* which is a product of Altova for modeling, editing, transforming, and debugging XML technologies. In the future, a specific GUI should be designed and developed, through which data can be input into the XML file and the data in the XML file can be edited, so that users do not have to handle the XML file directly. Or depending on how the system is implemented, the

GUI might also communicate with the database by eliminating the XML file in the middle.

Step 5: Build the design and analysis process. Since at the current stage, the design environment is integrated using a general purpose integration software package, *ModelCenter*, this step is specific to using *ModelCenter* as the integration tool.

Step 5a: Create wrappers, *ModelCenter* wrappers, for each analysis tool, either a disciplinary analysis tool or a mission analysis tool, so that it can communicate with the data management system. In *ModelCenter*, these wrappers enable each analysis tool to query data from and store data into the database through the data management system. If such wrappers have been created for some analyses programs, simply reuse them. The new wrappers should also be managed properly for later reuse.

Step 5b: Build the link between each analysis tool and the database through those wrappers created in Step 5a.

Step 5c: Model and build the whole design and analysis process based upon the dependency of the analyses programs. In *ModelCenter*, this is done through Scheduler, which can be Visual Basic Scripts. Given in Appendix I are the scripts for the design processes of both case studies. When we build such a process, the dependency of the analyses programs should be carefully considered. Sometimes, one analysis program (analysis program A) needs the information from

another analysis program (analysis program B). Therefore, in the design process, before we run analysis program A, analysis program B should be executed and the database should be updated with the new results from analysis program B. If there is no dependency among some of the analyses programs, they can be executed in parallel if possible.

Once these steps are finished, we will have database-supported design and analysis processes that can be executed automatically in *ModelCenter*.

Before the two case studies are described, it needs to mention that the data in Table 8, Table 9, Table 10, Table 11, Table 12, and Table 13 are based upon existing data. Using these data makes the comparison and justification of the results generated by the prototype integrated design environments easier for the author. In real world, depending on the requirements of a project, designers may not have the luxury to start a project with data of such precision. Some of the data in these tables are calculated based upon other attributes; for example, in Case Study I, the reference wing area in Table 10, the total horizontal tail area in Table 11, and the area of vertical tail in Table 12. It is the same for such variables of Case Study II.

7.2 Case Study I: A Notional Conventional Aircraft

This case study is performed to solve a simplified conceptual design problem of a notional conventional aircraft. It also serves the purpose of validating the mission analysis component and testing the command line commands provided by the data

management system. The problem to be solved is to size the aircraft in terms of a required mission, a given geometry configuration, and certain values of thrust loading and wing loading. It needs to be restated that the purpose of this case study is to investigate the efficacy of the proposed solutions to the research problems with a focus of validation and testing – not to solve such problems as to what principle or mechanism should be used for certain disciplinary analyses, how changes of the values of certain design variables affect the design results, what new technologies should be considered, etc. This notional aircraft is designed to be powered by two turbofan engines and have a payload of 1428 lb. The descriptions of the required mission, the geometry configuration, and disciplinary analyses involved are given next.

- Required mission

  The mission of this aircraft consists of warmup, takeoff, climb, subsonic cruise, constant speed turn, subsonic cruise, descend, loiter, and landing as shown in Figure 62. Please see Table 8 for a detailed description of each mission segment. For simplicity, it is assumed that descend and landing do not consume fuel.

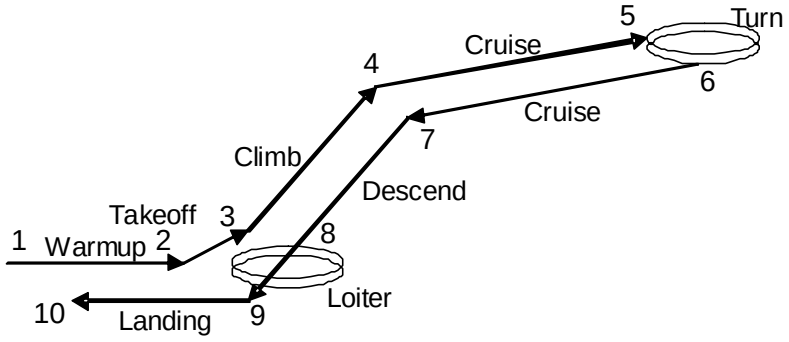

Figure 62. Case Study I: Mission of the Notional Conventional Aircraft

Table 8. Mission of the Notional Conventional Aircraft

Mission Segment		Conditions
1 ~ 2	Warmup	0ft, 2.2minutes, engine idle
2 ~ 3	Takeoff	0.8minutes, maximum power with afterburner
3 ~ 4	Climb	0->25000ft, maximum military power
4 ~ 5	Subsonic Cruise	0.95M, 25000ft ->26804ft, 225n.mi
5 ~ 6	Constant Speed Turn	0.85M, 26804 ft, 5min, maximum power with afterburner
6 ~ 7	Subsonic Cruise	0.95M, 26804->25000ft, 225n.mi
7 ~ 8	Descend	-
8 ~ 9	Loiter	0.37M, 20min
9 ~ 10	Landing	-

Figure 63. Case Study I: The Geometry Configuration of the Notional Aircraft [118]

- Geometry configuration

  The main body of this notional aircraft includes components of a fuselage, a wing, a horizontal tail, two vertical tails, a fuselage pod, and strakes, as depicted in Figure 63. Shown in Table 9, Table 10, Table 11, Table 12, and Table 13, are the details of some attributes of the geometric configuration of these components.

Table 9. Geometry Configuration of the Conventional Aircraft: Fuselage

Attributes	Value(s)	Unit
Fineness ratio of the nose	5.72	
Fineness ratio of the after body	2.25	
Length of the body	57.9	ft
Total fuselage fineness ratio	10.07	
Maximum diameter of the body	6.41	ft

Table 10. Geometry Configuration of the Conventional Aircraft: Wing

Attributes	Value(s)	Unit
Sweep angle	20.23	deg
Aspect ratio	3.62	
Taper ratio	0.36	
Thickness-to-chord ratio at root	0.07	
Thickness-to-chord ratio at tip	0.04	
Fraction of chord covered by leading edge flap	0.12	
Fraction of chord covered by trailing edge flap	0.15	
Fraction of chord covered by trailing edge flap	0.15	
Dihedral angle	-2	deg
Position of wing quarter chord at wing root	0.52	
Reference wing area	451.12	ft^2
Wing twist	3.00, 0.00, 0.00	deg
Chord covered by leading edge	0.12	
Chord covered by trailing edge	0.15	
Position of 1/4 chord at wing root centering given as a fraction of body length	0.52	

Table 11. Geometry Configuration of the Conventional Aircraft: Horizontal Tail

Attributes	Value(s)	Unit
Sweep angle	52.29	deg
Aspect ratio	2.72	
Taper ratio	0.34	
Thickness-to-chord ratio at root	0.06	
Thickness-to-chord ratio at tip	0.04	
Position of trailing edge of root chord	0.94	
Total horizontal tail area	173.43	ft^2

Table 12. Geometry Configuration of the Conventional Aircraft: Vertical Tail

Attributes	Value(s)	Unit
Sweep angle	43.40	deg
Aspect ratio	2.46	
Taper ratio	0.34	
Thickness-to-chord ratio at root	0.05	
Thickness-to-chord ratio at tip	0.03	
Position of trailing edge of root chord	0.87	
Area of one tail	145.82	ft^2
Number of vertical tails	2	

Table 13. Geometry Configuration of the Conventional Aircraft: Strake

Attributes	Value(s)	Unit
Sweep of the strake leading edge	79.97	deg
Ratio of the x location of the strake intersection with the nose along the center line to the total body length	0.21	
Ratio of strake span/2 to total span/2	0.23	

- Disciplinary Analyses

  Disciplinary analyses needed in this conceptual design include optimization, aerodynamics, atmospheric properties, engine cycle, and geometry design. The mission analysis component needs information from the above mentioned analyses programs. For simplicity and in order to validate the design results against those from ACSYNT, which is well trusted, the aerodynamic analysis program, the engine cycle analysis program, and the geometry design program used for the design are decomposed components of ASCYNT for corresponding disciplinary analyses.

The amount of information needed and generated by these disciplinary analyses together with mission analyses is large. Besides the input data about mission and geometry configuration given previously, there are also physics attributes, for example, those data of drag polar and engine thrust. They are managed by the data management system in the design environment.

7.2.1 Integrated Design Environments

For easy comparison, the design environment is integrated in two versions, one of which is supported by the data management system while the other one is not. These environments are illustrated in Figure 64 and Figure 65, both of which provide magnified views of the mission analysis component that consists of mission analyses programs for warmup, takeoff, climb, cruise, descend, loiter, and landing. As mentioned previously, it is assumed that descend and landing do not consume fuel in order to simplify the problem. Disciplinary analysis tools integrated into the design environments for this case study include geometry, aerodynamics, propulsion, and atmosphere. All the analyses programs can be distributed on different computers at different locations. This is quite important for enabling designers to work collaboratively around the world and for the effective utilization of computing resources of both software and hardware. With this design environment, designers are given the flexibility to choose different analysis tools based on their availability and the required level of analysis fidelity. For example, the aerodynamic analysis tool integrated in these design environments, which is a component decomposed from ACSYNT, is at the conceptual design level. If higher fidelity

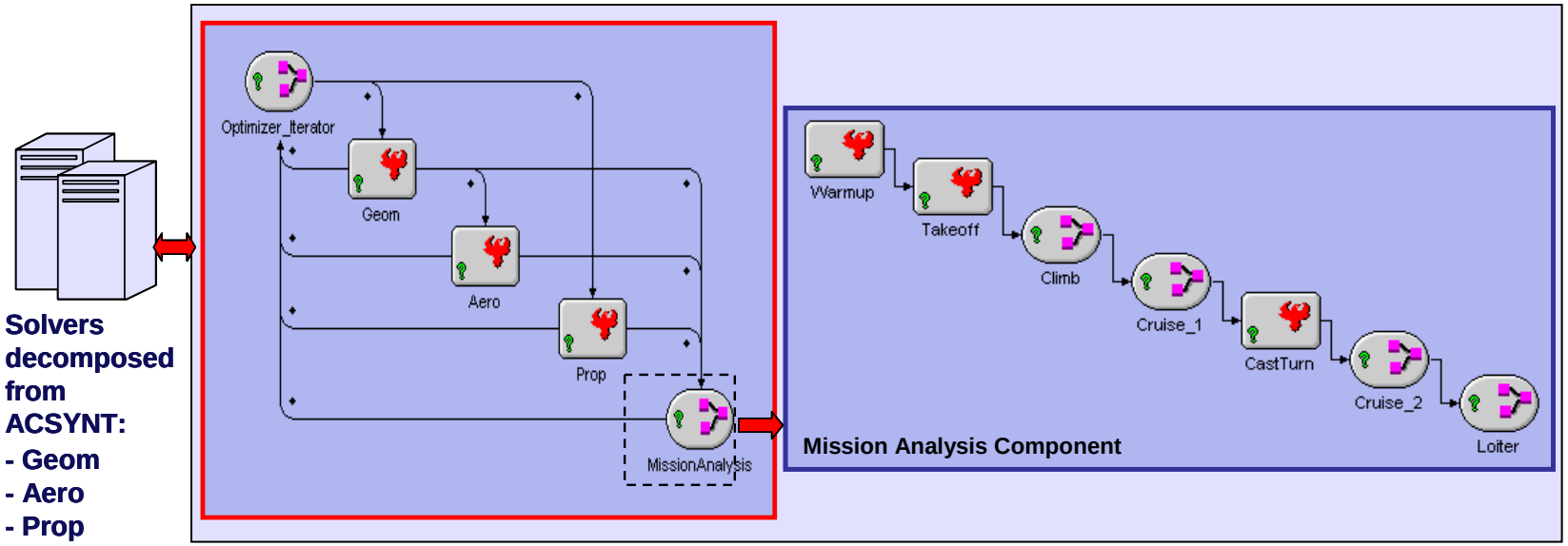

Figure 64. Case Study I: Integrated Design Environment without the Support of the Data Management System

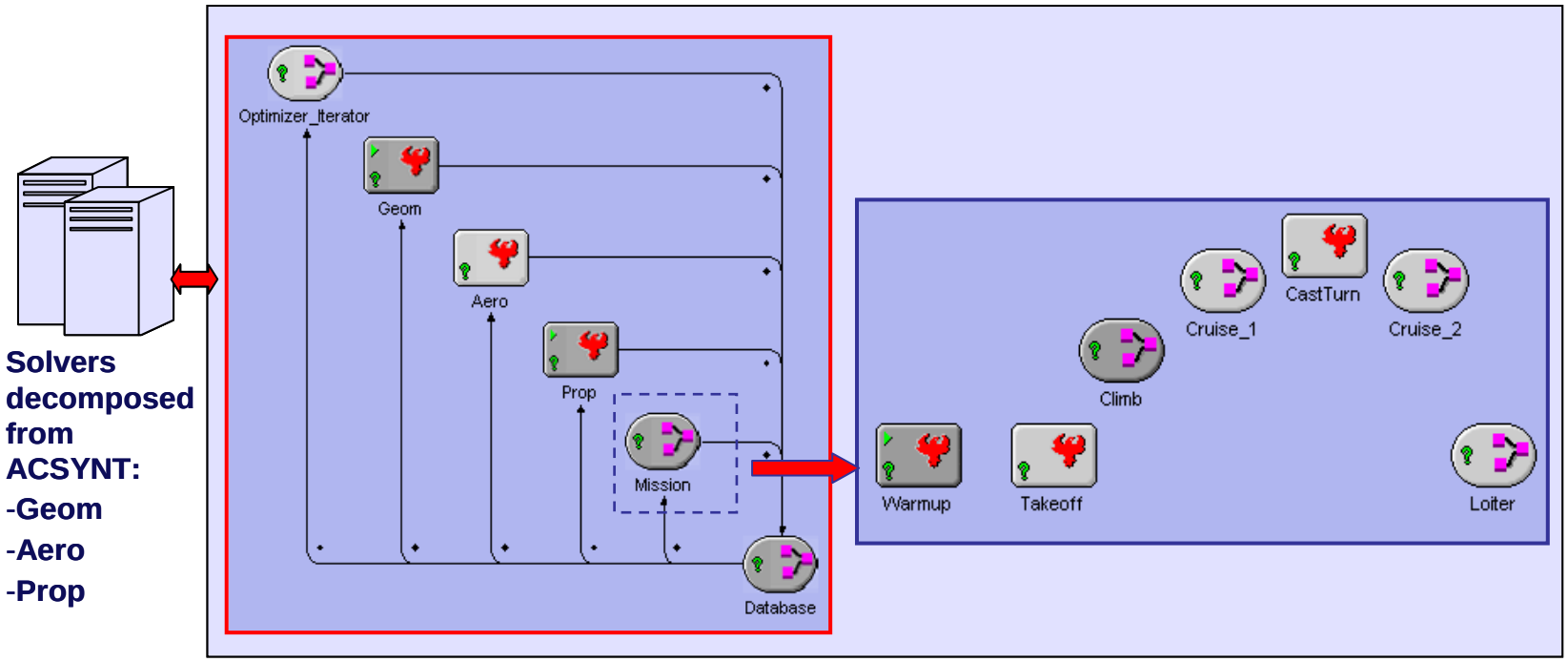

Figure 65. Case Study I: Integrated Design Environment with the Support of the Data Management System

aerodynamic analyses need to be conducted, other aerodynamic analysis tool(s) can be used to replace the one integrated into the environments. The drawback of this important benefit is the writing of specific wrappers for each analysis tool. However, these wrappers can be reused. For easy access and reuse, they can be organized into libraries of disciplinary analyses tools and mission analyses tools.

In these conceptual design environments, each building block is wrapped as an object. In the design environment without the support of the data management system shown in Figure 64, these objects are connected with each other. They need to communicate with all the other objects that pass information to them or rely on their analysis results. As has been explained in Chapter 5, in such an environment, interoperability is poor among the analysis programs. For example, if the aerodynamic analysis tool integrated into the design environment needs to be replaced by a higher fidelity analysis program, wrappers should be written for the new analysis tool as well as other analysis tools which are already integrated into the design environment in order to build up the communication among them. The worst-case scenario is that the design process might need to be reconstructed.

However, it can be seen that in the design environment supported by the data management system, such objects are standalone with each other at runtime; each of them only communicates with the database; each time when the design processes are invoked to execute these objects, they query the database for the input data they need first and then stores results into the database once the executions are finished. If an analysis program needs be swapped out and/or a new analysis tool needs to be integrated, it can be done through a wrapper that builds up the communication with the database only.

Interoperability among the analyses programs is improved dramatically. In a sense, it is the divide-and-conquer approach to solve a complicated problem. At the development stage, each of the tools integrated into the design environment is in general dependent indirectly on the others in that the central data management system must support any information they share and must do so at the a compatible level of fidelity. This share of information must be managed when a new tool is added or an existing tool is changed. As described in section 6.1, the integration of the database with other analyses programs is achieved through *ModelCenter* wrappers. These wrappers enable the analyses programs to query data from and store data into the database through the data management system. Listed in Appendix J are these wrappers.

The same is true for the relationships among other disciplinary analysis tools. The mission segment objects in the integrated design environment are independent with each other at run time. This enables designers to construct a wide range of types mission profile easily. For example, a conventional takeoff mission segment can be replaced at runtime using a vertical takeoff segment without affecting other mission segment objects. In these integrated design environments in *ModelCenter*, it can be done by simply dragging the mission segment objects needed from the mission segment object "library", dropping them to the work area, and then connecting them with the database. Further, in these design environments, designers have the capability to amend the mission analysis for desired accuracy, i.e., the mission segment can be divided into as many legs as needed in order to achieve a certain accuracy. For example, in this case study, the climb segment is divided into ten legs. Please see Appendix H for the screen shots of the integrated design environments in *ModelCenter*.

Sometimes, one program needs input from other programs, for example, some of the aerodynamic analysis inputs are the outputs of geometry design; and in this case, the geometry design component should be executed before the aerodynamic analysis component. At this research stage, such dependency is captured in the design processes. In the future, research needs to be conducted on how to combine such information with design data and manage it using databases. The conceptual design process of this notional aircraft can be found in Appendix I. It is a Visual Basic script, called Scheduler in *ModelCenter*.

7.2.2 Validation of the Mission Analysis Component and Design Result

Since the mission analysis component in the prototype integrated design environments presented in the previous section is built from scratch, its feasibility and validity should be tested. Due to the fact that in the design environments the disciplinary analyses programs used for the components of "*AERO*", "*PROP*", and "*GEOM*" are decomposed from ACSYNT, the design and analysis results from ACSYNT for the same design concept are used for comparison. It should be noted that the design and analysis results generated in the two integrated design environments should be the same because the only difference between these two design environments lies in the fact that one is supported by the data management system and the other is not. The integration of the data management system in the design environment helps to simplify the design process and improve data

sharing among analyses programs; however, it should not affect the design and analysis results.

The primary design and analysis results from both of the new integrated object-oriented design environments and ACSYNT are shown in Table 14 and Table 15. When a comparison is to be made, we should set the design and analysis conditions equally so that we can reach reasonable conclusions. Following is a list of these conditions:

- The disciplinary analysis programs used in the new design environments are the same as those in ACSYNT. They are the decomposed components from ACSYNT.
- The design and analyses are conducted based on the same mission described in Table 8.

Table 14. Case Study I: Primary Result Comparison

Weight (pound)	Calculated Results:			
	ACSYNT	**NextADE**	**Absolute Difference**	**Relative Difference**
Takeoff Gross Weight	35670	35642	28	0.0785%
Mission Fuel Weight	10918	10897	21	0.1933%

** To get these results, the empty weight and the payload weight of the aircraft are set to be the same amount, which equals to 22772 lb and 1428 lb respectively, for both ACSYNT and the NextADE in order to make the comparison baseline the same.*

Table 15. Case Study I: Required Mission Fuel Weight Comparison

Mission Segment		Calculation Results: Required Fuel Weight of Each Segment			
		ACSYNT	NextADE	Absolute Difference	Relative Difference
1 ~ 2	Warmup	65.20	65.00	0.20	0.31%
2 ~ 3	Takeoff	778.50	776.30	2.20	0.28%
3 ~ 4	Climb	621.20	585.60	35.60	5.73%
4 ~ 5	Subsonic Cruise Climb	2809.00	2787.30	21.70	0.77%
5 ~ 6	Constant Speed Turn	2863.70	2861.40	2.30	0.08%
6 ~ 7	Subsonic Cruise Climb	2754.40	2799.20	44.80	1.63%
7 ~ 8	Descend	0.00	0.00	0.00	-
8 ~ 9	Loiter	1026.50	1022.10	4.40	0.43%
9 ~ 10	Landing	0.00	0.00	0.00	-

- The design and analyses are conducted based on the same geometry configuration described in Section 6.1.
- The vehicle is powered by the same turbofan engines for the analyses of both ACSYNT and the integrated design environments.
- Thrust loading and wing loading of the vehicle are set to equal 0.59 and 79.07 lb/ft^2 respectively for both ACSYNT input file and the new integrated design environments.
- The purpose of mission analysis is to calculate fuel weight required to accomplish the required mission. The change of fuel weight will in turn change the total takeoff gross weight. The total takeoff gross weight is a summation of fuel weight, payload weight, and empty weight. In order to focus on the

difference between the fuel weight calculated using the new mission analysis component and that from ACSYNT, the empty weight and the payload weight of the aircraft are set to be the same, which is equal to 22772 lb and 1428 lb respectively, for both ACSYNT and the new integrated design environments. The discrepancy (seven pounds) between the two values of absolute difference of total takeoff gross weight and mission fuel weight is caused by the optimization algorithms used and how trapped fuel weight is modeled. At this stage, two options for empty weight calculation are actually available in the new sizing environment. One is to fix the empty weight when it is a known non-varying value. The other option is to vary the empty weight based on historical data regression. In this example, empty weight is fixed. However, for some unconventional concept vehicles, it will remain to be a problem that needs to be solved since historical data are not available for the regression analysis. One solution to this problem is to incorporate an empty weight calculation program as a disciplinary analysis object, which calculates the empty weight using geometry data and other information such as materials.

Additionally, the mission analysis component continues to track well as key parameters are varied, such as the block range as indicated in Figure 66. The data in Figure 66 indicate that the new integrated design environment and ACSYNT generate reasonably close vehicles over a wide mission range. Based upon the above comparison, the conclusion can be made that the new object-oriented mission analysis component, which is redesigned and developed based dissertation concepts, produces satisfactory results.

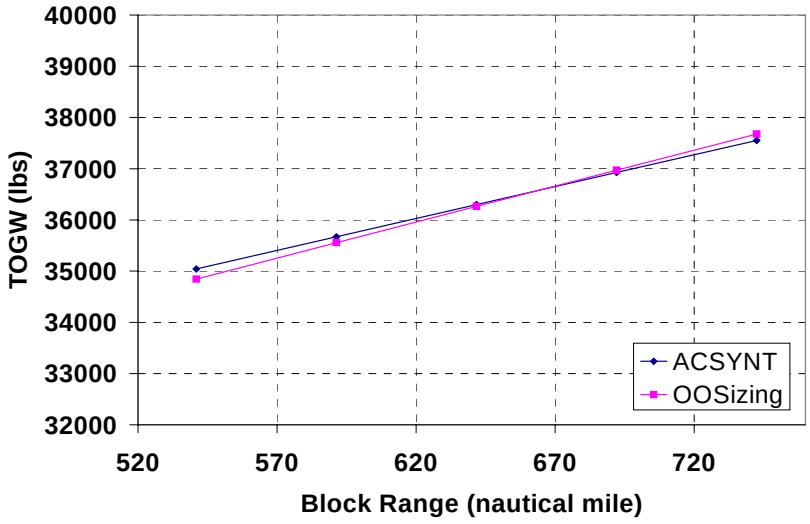

Figure 66. Case Study I: Comparative Results with Variable Block Ranges

7.2.3 Command Line Commands Test

The tables in the database described in section 5.3.2.1 have been populated with the conceptual design data of the notional conventional aircraft. The amount of data is large. Obviously, we can query and manage these data directly in the MySQL environment using MySQL commands. However, for designers, this is a daunting task due to the large amount of data and the complexity of the relationships, especially for the fact that they most likely do not know how these data are stored in the database. Thus, the data management system should provide certain functionalities so that the database can be queried and managed in an easy way. On the other hand, by isolating the database from end users, this is actually also an effective way to protect the database itself and avoid the unnecessary problems of maintenance.

As shown previously, Table 7 has example commands for achieving such functionalities. In order to show how these commands work, a few snapshots of the execution results of some of these commands will be provided in this section. These snapshots are for the purpose of demonstration. Data shown in these snapshots are only a small portion of the data stored in the database - not every attribute about a data object is listed in the snapshots. These example commands are chosen with the intention that the specified geometry component or the specified analysis has the appropriate amount of data so that the result won't take much paper space. However, they can still clearly show how these commands work.

- Query the project(s) store in the database:

```
D:\OAD\jar>java -jar oad.jar -l
OAD: [Project list]
OAD: OADTest
```

  The results show that there is one project stored in the database, which is named as "OADTest".

- Query a specific project:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest
OAD: [Project]
OAD: name:OADTest
OAD: desc:This is an implementation example.
OAD: [Requriement list]
OAD: missionProfile
OAD: [DesignConcept list]
OAD: OAD01
```

  The results show that there is one set of requirements named "missionProfile" and one design concept called "OAD01" for project "OADTest".

- Query a specific set of requirements:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest -r missionProfile
OAD: [Requirement]
OAD: name:missionProfile
OAD: desc:the mission for the aircraft
```

```
OAD: [Variable list]
OAD: ALTITUDE_loiter=0.00
OAD:    desc: the altitude of loiter
OAD:    unit: ft
OAD: ALTITUDE_takeoff=0.00
OAD:    desc: the altitude of takeoff
OAD:    unit: ft
OAD: ALTITUDE_turn=26804
OAD:    desc: the altitude of turn
OAD:    unit: ft
OAD: ALTITUDE_warmup=0.00
OAD:    desc: the altitude of warmup
OAD:    unit: ft
OAD: CLIMBSPEED=300.0
OAD:    desc: the speed of climb
OAD:    unit: ft/sec
OAD: CRUISEMACH_cruise1=0.95
OAD:    desc: the crusie mach number of the first cruise segment
OAD:    unit:
OAD: CRUISEMACH_cruise2=0.95
OAD:    desc: the mach number of the second cruise segment
OAD:    unit:
OAD: DURATION_turn=300
OAD:    desc: the duration of turn
OAD:    unit: sec
OAD: ENDURANCE=1200
OAD:    desc: the endurance of loiter
OAD:    unit: sec
OAD: FINALALTITUDE_climb=25000
OAD:    desc: the final altitude of climb
OAD:    unit: ft
OAD: FINALALTITUDE_cruise1=26804
OAD:    desc: the final altitude of the first cruise segment
OAD:    unit: ft

……

OAD: INITIALALTITUDE_cruise2=26804
OAD:    desc: the initial altitude of the second cruise segment
OAD:    unit: ft
OAD: LOITERMACH=0.37
OAD:    desc: the mach number of loiter
OAD:    unit:
OAD: RANGE_cruise1=1367128
OAD:    desc: the range of the first cruise segment
OAD:    unit: ft
OAD: RANGE_cruise2=1367132
OAD:    desc: the range of the second altitude
OAD:    unit: ft
OAD: TAKEOFFTIME=48.0
OAD:    desc: the duration of take off
OAD:    unit: sec
OAD: TURNMACH=0.85
OAD:    desc: the mach number of turn
OAD:    unit:
OAD: WARMUPTIME=132
OAD:    desc: the duration of warmup
OAD:    unit: sec
```

The variable list shown in the results are those variables related to the requirement set of “missionProfile”. These variables are listed in an alphabetic order.

- Query a specific design concept:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest -d OAD01
OAD: [DesingConcept]
OAD: name:OAD01
OAD: desc:This is a design concept for the project of OADTest.
OAD: [GeometryConfiguration list]
OAD: Geom01
OAD: [Analysis list]
OAD: warmup
OAD: loiter
OAD: castTurn
OAD: aerodynamicAnalysis
OAD: cruise2
OAD: cruise1
OAD: takeoff
OAD: climb
OAD: optimization
OAD: propulsionAnalysis
```

The results show that design concept “OAD01” has one geometry configuration named “Geom01”, and needs disciplinary analyses of “optimization”, “aerodynamicAnalysis”, “propulsionAnalysis”, and mission analyses including “warmup”, “takeoff”, “cruise”, “loiter”, etc.

- Query a specific geometry configuration:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest -d OAD01 -g Geom01
OAD: [GeometryConfiguration]
OAD: name:Geom01
OAD: desc:The configuration includes the components of wings,
horizontal tail vertical tail, fuselage, fuselage mounted pod, and
two engines. The engines are existing engines. Detail info about
the enignes are not included.
OAD: [SystemComponent list]
OAD: cockpit
OAD: engine
OAD: wing
OAD: horizontal_tail
OAD: strake
OAD: fuselage_pod
OAD: vertial_tail
OAD: fuselage
```

The results show that geometry configuration “Geom01” consists of components of “wing”, “engine”, “fuselage”, “horizontal tail”, etc.

- Query a specific system component:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest -d OAD01 -g Geom01 -c
wing
OAD: [SystemComponent]
OAD: name:wing
OAD: desc:Geometry information of the wing. One wing.
OAD: [Variable list]
OAD: ARW=3.62
OAD:    desc: aspecr ratio, wing
OAD:    unit:
OAD: DIHED=-2.00
OAD:    desc: wing dihedral
OAD:    unit: deg
OAD: FDENWG=50.00000
OAD:    desc: density of fuel in the wing
OAD:    unit: lb/f^3
OAD: FIXWOS=79.069957
OAD:    desc: fixed wing loading
OAD:    unit:
OAD: LFLAPC=0.12
OAD:    desc: L.E. flap CHORD/(wing CBAR) - chord covered by
leading edge
OAD:    unit:
OAD: SWFACT_wing=0.9877000
OAD:    desc: wetted area multiplier
OAD:    unit:
OAD: SWING=450.8679
OAD:    desc: reference area, wing
OAD:    unit: ft^2
OAD: SWPW=20.23314
OAD:    desc: wing reference swept
OAD:    unit: deg
OAD: TCWR=0.07
OAD:    desc: root T/C, wing
OAD:    unit:
OAD: TCWT=0.039999999
OAD:    desc: tip T/V, wing
OAD:    unit:
OAD: TFLAPC=0.150
OAD:    desc: T.E. flap CHORD/(wing CBAR) - chord covered by
trailing edge
OAD:    unit:
OAD: TRWING=0.36
OAD:    desc: taper ratio, wing
OAD:    unit: ft
OAD: TWISTW=3.000000, 0.000000, 0.000000
OAD:    desc: twist of wing
OAD:    unit: deg
OAD: WFFRAC=0.30
OAD:    desc: available fuel volume in wing that is to be filled
with fuel
OAD:    unit:
OAD: XWING=0.5200000
OAD:    desc: position of 1/4 chord at wing root center line given
as a fraction of body length
```

```
OAD:    unit:
OAD: ZRTWG=0.00
OAD:    desc: root elevation, wing
OAD:    unit: ft
```

Shown in the results are the variables related to the “wing” component. And these variables are listed in an alphabetic order.

- Query a specific analysis:

```
D:\OAD\jar>java -jar oad.jar -l -p OADTest -d OAD01 -a cruise1
OAD: [Analysis]
OAD: name:cruise1
OAD: desc:
OAD: [Variable list]
OAD: CRUISEMACH_cruise1=0.95
OAD:    desc: the cruise mach number of the first cruise segment
OAD:    unit:
OAD: FINALALTITUDE_cruise1=26804
OAD:    desc: the final altitude of the first cruise segment
OAD:    unit: ft
OAD: FINALWEIGHT_climb=34209.9931262486
OAD:    desc: the final weight of climb
OAD:    unit: lb
OAD: FINALWEIGHT_cruise1=31423.167263672
OAD:    desc: the final weight of the first cruise segment
OAD:    unit: lb
OAD: INITIALALTITUDE_cruise1=25000
OAD:    desc: the initial altitude of the first cruise segment
OAD:    unit: ft
OAD: RANGE_cruise1=1367128
OAD:    desc: the range of the first cruise segment
OAD:    unit: ft
```

Shown in the results are the variables related to “cruise1” analysis, which is the analysis for the first cruise segment, in alphabetical order. In the results, both inputs, such as altitudes, and outputs, such as weights, of the “cruise1” analysis are listed.

7.3 Case Study II: An Unconventional Aircraft

Domain flexibility and analysis scalability are two important issues that have been addressed when the proposed framework of the NextADE was formulated in Chapter 4. In order to demonstrate these capabilities, a future unconventional aircraft concept is chosen for this case study. This unconventional aircraft is a derivation based upon an existing aircraft - Cessna 206H Stationair. Unlike the Cessna 206H Stationair, which is powered by a traditional piston engine, the derivation concept is powered by a hybrid propulsion system, which is a combination of a fuel cell engine and a traditional piston engine. This derivation concept will be called Cessna Hybrid.

Under environmental and social economical pressures, over the last decade, people have been trying to find different energy sources other than hydrocarbon fuels to power aircraft. Fuel cells, as one of the energy alternatives, which include fuel cells, atomic energy, solar energy, beamed microwave energy, etc, drew researchers' attention due to the fact that they were used to power electrical systems of spacecraft on the space flights of NASA [119]. To design aircrafts that are powered by these unconventional energy sources, one of the obstacles that designers face is that traditional aircraft sizing methods introduced in Chapter 3 are not immediately applicable. Nam et al in Aerospace System Design Laboratory (ASDL) of Georgia Institute of Technology have conducted research on formulating a generalized aircraft sizing method that is applicable to both conventional and unconventional aircrafts powered by different energy sources [120, 121]. The proposed sizing method allows system design engineers to evaluate the impact of alternate energy sources and new propulsion concepts on aircraft integration.

Nevertheless, traditional aircraft conceptual design programs being used in practice, which were listed in Chapter 3, were developed based upon traditional aircraft conceptual design methods. They cannot be used for the design of such unconventional vehicles and only provide limited capabilities for these advanced research efforts. However, the proposed object-oriented distributed design framework for the NextADE can be used to design such vehicles; it also provides the flexibility for related advanced research activities. Before demonstrating these capabilities, it will be helpful to restate the two major reasons why tradition conceptual design programs are not good enough to meet the challenges of designing the Cessna Hybrid:

- Domain flexibility: A fuel cell engine is a type of engine consuming electric energy converted from the hydrogen in the fuel cell; unlike traditional dissipative fuel, the fuel weight remains on board partially or totally. None of the design algorithms hard coded into the traditional conceptual design programs, which are based upon dissipative fuel, can handle the design of such a vehicle properly.
- Analysis scalability: The built-in propulsion system design components of the traditional conceptual design programs are good at modeling traditional propulsion system such as piston engines, turbo engines, etc. They are neither suitable nor easily modifiable to model the hybrid propulsions system of the Cessna Hybrid, which is basically a combined utilization of a fuel cell and a battery system and a piston engine. The detailed configuration of this hybrid propulsions system will be introduced next.

The Cessna 206H Stationair, as shown in Figure 67, is a six-seat utility transport aircraft. It is one of the models of the Cessna aircraft family owned by the Cessna Aircraft Company [122], which is a subsidiary of Textron Inc. [123]. The mission of the Cessna Hybrid is the same as the mission of the Cessna 206H Stationair, which is illustrated in Figure 68 and described in detail in Table 16. The mission consists of segments of warmup, takeoff, climb, cruise, loiter, descend, and landing. Besides the mission, listed in Table 17 are some of the published specifications of the aircraft from reference [124].

Figure 67. Cessna 206H Stationair [125]

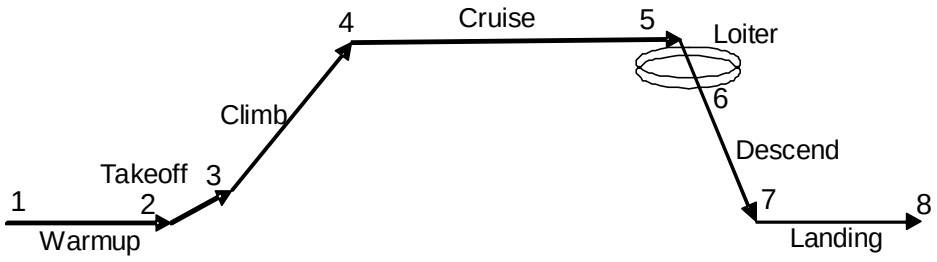

Figure 68. Case Study II : Mission of the Unconventional Aircraft

Table 16. Mission of the Unconventional Aircraft

Mission Segment		Conditions
1 ~ 2	Warmup	0ft, 5.0minutes
2 ~ 3	Takeoff	Oft, 1.8minutes
3 ~ 4	Climb	0->6200ft, maximum military power if the piston engine is used
4 ~ 5	Cruise	0.22M, 6200ft, 605n.mi
5 ~ 6	Loiter	0.2M, 6200 ft, 50min
6 ~ 7	Descend	6200 -> 0 ft
7 ~ 8	Landing	-

Table 17. Some Specifications of Cessna 206H Stationair [119]

External Dimensions:	
Wing span	36 ft
Wing chord at tip	3 ft 8.5 in
Wing chord at root	5 ft 4 in
Wing aspect ratio	7.4
Length overall	28 ft 3 in
Height overall	9 ft 3.5 in
Areas:	
Wing, gross	175.5 ft^2
Ailerons, total	17.35 ft^2
Trailing edge flap, total	28.85 ft^2
Fin	11.62 ft^2
Rudder, incl tab	6.95 ft^2
Tailplane	24.84 ft^2
Elevators, incl tab	20.08 ft^2
Weight and Loading:	
Weight, empty	2176 lb
Max fuel	528 lb
Max ramp weight	3614 lb
Max wing loading	20.51 lb/ft^2
Max power loading	12 lb/hp
Performance:	
Max level speed at sea level	174 mph
Cruising speed	174 mph
Max rate of climb at sea level	988 ft/min
Range with max fuel, 45 min reserve at 6200ft	605 n.mi.

An aircraft powered by a hybrid propulsion system is still not feasible with the state of the art technologies. However, as stated previously it is very promising in the future and research is being conducted in this direction. The proposed object-oriented distributed framework provides a powerful design environment for such research activities. The hybrid propulsion system used to power the Cessna Hybrid is an advanced concept. It consists of 1) a traditional six-cylinder piston engine, which consumes a dissipative fuel power source, 2) a hydrogen fueled proton exchange membrane fuel cell (PEMFC) with brushless DC electric motor, which has the options of dissipative or non-dissipative fuel power sources, and 3) a lithium polymer battery with brushless DC electric motor, which consumes a non-dissipative fuel power source. Shown in Figure 69 is the configuration of the propulsion system.

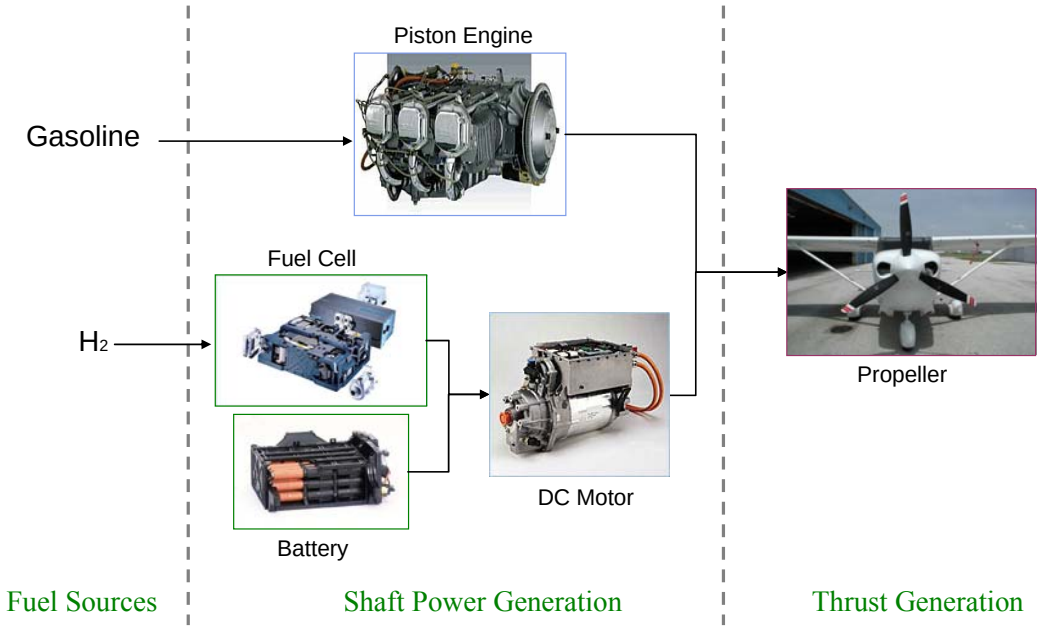

Figure 69. Case Study II: Configuration of the Hybrid Propulsion System [126]

To model this hybrid propulsion system, several analyses programs are needed. A decomposed propulsion module from ACSYNT is used for the design and analysis of the piston engine that consumes traditional dissipative fuel. The PEMFC is modeled with a physics-based model of a hydrogen processing PEMFC. For the fuel cell system, performance data is calibrated against Ballard Fuel Cells and weight models are constructed from the published data [127]. A future technology carbon nanotube substrate (CNS) is used in order to reduce weight. The CNS acts as a sponge to hold the hydrogen fuel with an efficiency of 65%. The battery model simply considers the weight of the Li-Poly battery based on the required energy, and ensures that the battery is large enough for the required power. Also, each of the electrical power plants is coupled with a brushless DC motor to produce shaft power. A first order model is used, and weight is calculated based on regressions of data collected for existing brushless DC motors. Illustrated in Figure 70 is the weight calculation of this hybrid propulsion system.

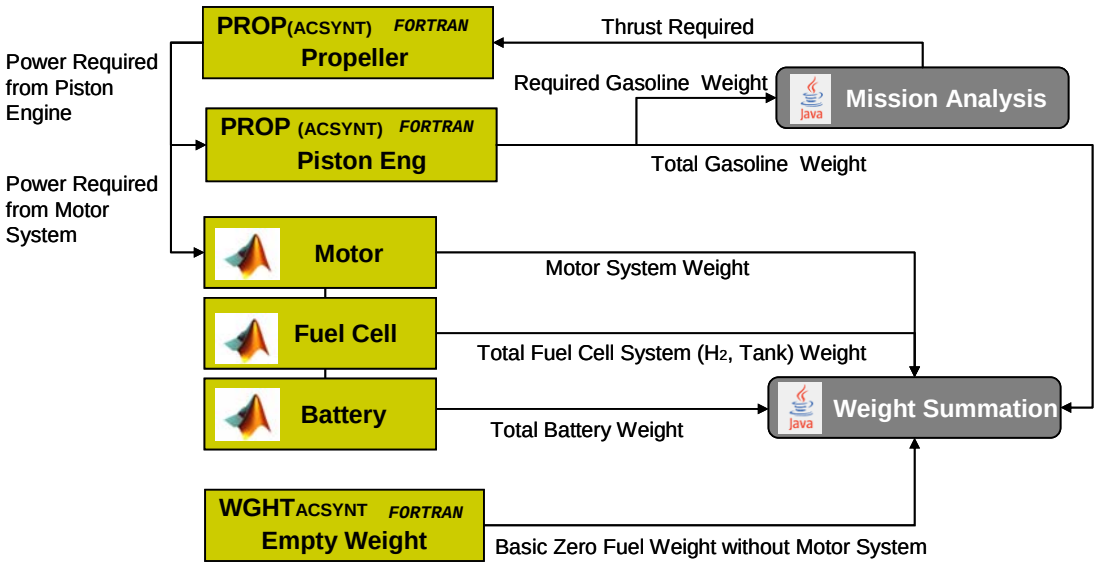

Figure 70. Case Study II: Weight Calculation of the Hybrid Propulsions System [126]

7.3.1 Integrated Design Environments

The same as Case Study I, *ModelCenter* is used as the integration tool to construct the integrated conceptual design environments based upon the proposed framework for the case study of the Cessna Hybrid. The procedure described in Section 7.1 is followed. Illustrated in Figure 71 and Figure 72 are the integrated design environments with and without the support of the data management system.

In these environments, disciplinary analyses modules decomposed from ACSYNT are used for the aerodynamic analysis, geometry design, and weight analysis. Besides these programs, as illustrated in the previous section, to design the Cessna Hybrid, we also need analyses programs to model the hybrid propulsion system, which includes programs for the modeling and analyses of the piston engine, the fuel cell system, the motor, and the battery. The programs used to model the fuel cell system, the motor, and the battery are *Matlab* scripts developed by Won [126]. As mentioned previously, a decomposed propulsion analysis module from ACSYNT is used to model the piston engine.

Due to the fact that the conceptual design algorithms for aircrafts consuming dissipative fuel and non-dissipative fuel are different, new mission analysis classes accommodating the corresponding different treatments of fuel weight on board are written and added to the mission analysis library. These new mission analysis classes can be used for dissipative, non-dissipative, and semi-dissipative cases to fulfill the needs of designing Cessna Hybrid. This is done through a variable called DFP (dissipative fuel percentage), which specifies the distribution of power between the dissipative and non-

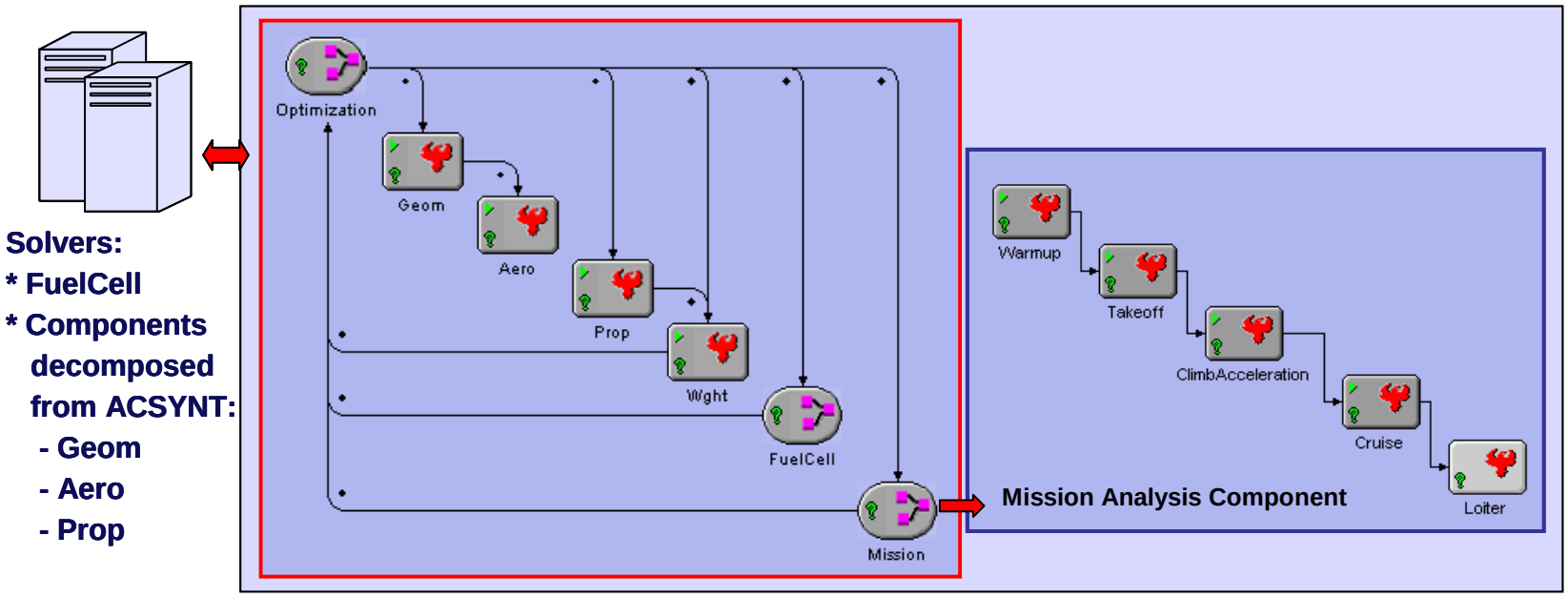

Figure 71. Case Study II: Integrated Design Environment without the Support of the Data Management System

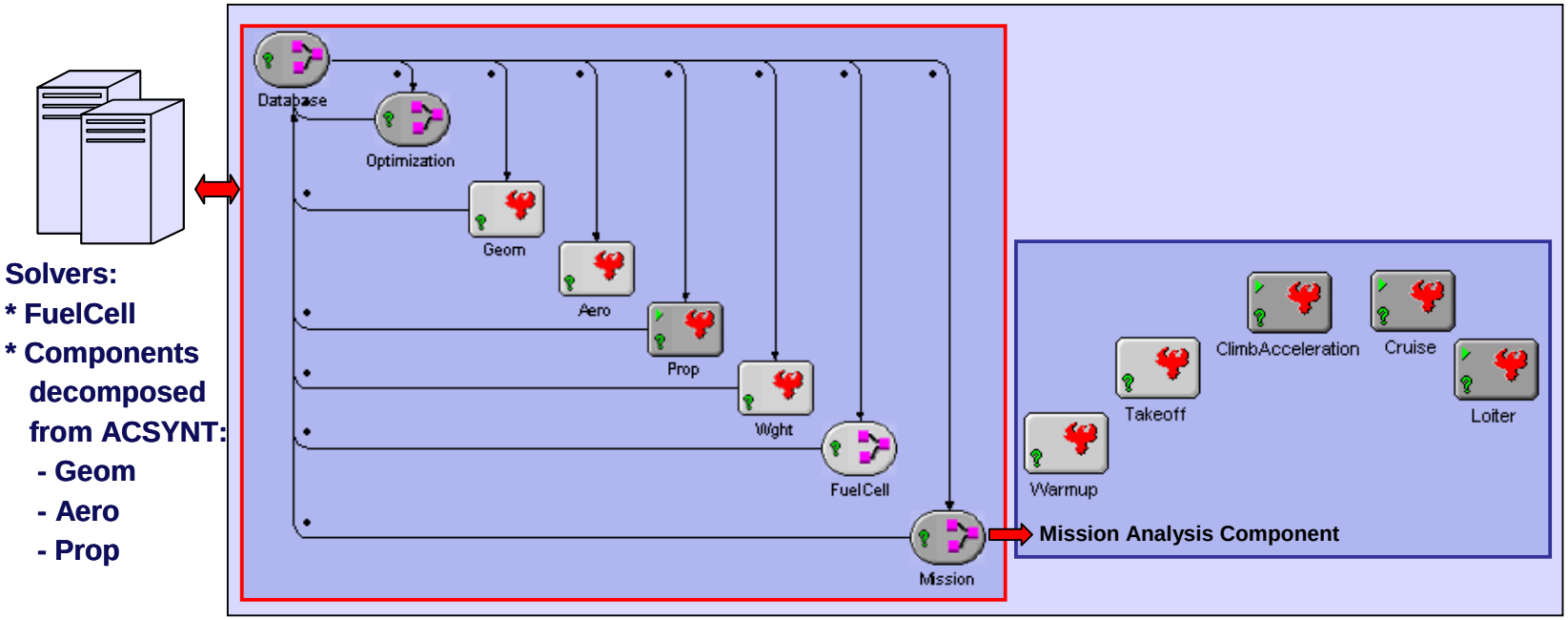

Figure 72. Case Study II: Integrated Design Environment with the Support of the Data Management system

dissipative fuel sources. It is assumed that, for one design scenario, the power distribution settings are the same in all the mission segments, except segments that use idling (e.g. warmup, descend) and max power (e.g., takeoff, climb). Together with the other disciplinary analysis programs integrated into the design environment and the design process, a first order modeling and analysis environment was constructed for the Cessna Hybrid.

The proposed object-oriented design environment provides designers flexibility to add more functionality easily as needed without affecting other building blocks for that the central data management system eliminates the direct dependency among them. However, using the traditional design programs, this might be a daunting task as discussed previously. To accomplish this using the traditional conceptual design programs, designers need to first dig into the source codes in order to understand them. This task itself might be a difficult one due to the fact that the source codes might not be available to designers or they might not be well documented. After this step, designers need to rewrite the codes. Sometimes, the whole architecture of the program needs to be changed. The amount of work might be equivalent to, or more, than that of designing and developing a whole new design program.

Compared to the two integrated design environments, the interoperability of the one supported by the central database management system is much better than the one without the support. The utilization of the database management system enables the analysis programs, which are wrapped as objects, to act independently. They only communicate with the database, querying input from the database and storing the output into the database after finishing execution. Without the support of the database

management system, the communication of one analysis program with any other analysis program needs to be taken care. For example, each of the mission analysis objects needs the information from the aerodynamic analysis object as inputs. The direct communication channel should be built for each of these dependencies. Doing so, each of the wrappers of the mission analysis objects should be customized according to the inputs/output details of the aerodynamic analysis object. If the integrated aerodynamic analysis program needs to be replaced with a different one, all the wrappers should be modified and new wrappers should be written; all the dependent relationships need to be rebuilt.

However, with the support of the database management system, these analysis objects are more isolated with each other; the changes of one analysis program won't affect others as much. For example, if Analysis A uses attribute "diameter", which is an output of Analysis B, as an input, but the input for a later version of Analysis A is somehow changed to be "radius". In this situation, we can solve the problem by either updating the wrapper of Analysis A by converting "diameter" queried from the database to "radius" or changing the wrapper of Analysis B by converting "radius" to "diameter" before it stores it into the database. We do not have to change the database structure and Analysis A or Analysis B. This is the desired trait for variable fidelity analysis. If one of the analysis programs needs to be replaced with a different one, having a higher or lower fidelity depending on the design requirement, designers simply swap them. The database management system works as a central translator among them. The interoperability of the system is thus improved; at the same time, the procedure to build the design process is simplified.

Because of the improved domain flexibility, analysis scalability, and interoperability, the integrated design environment based upon the proposed framework is fully capable to meet the requirements of designing unconventional aircraft concepts. In the following section, the design and analysis results of Cessna Hybrid from the integrated design environments will be listed and discussed.

7.3.2 Results of Study

In the prototype integrated design environment supported by the data management system, for validation purposes, design and analysis of the Cessna 206H Stationair is performed first. The results summarized in Table 18 and Table 19 indicate that Cessna 206H baseline specifications (powered by a traditional piston engine only) are matched within 1% for the baseline mission described in Table 16. In Case Study I, in order to validate the mission analysis component, the empty weight of the aircraft was given. However, as shown in Table 18, the empty weight here is not an input but an output.

The implementation of the hybrid propulsion system, where the power is distributed between the PEMFC and the piston engine, is then conducted for the study of the Cessna Hybrid. Results of this study for different DFP values are plotted in Figure 73, Figure 74, Figure 75, and Figure 76.

Table 18. Case Study II: Weights of the Baseline Concept

Takeoff Gross Weight	3619 lbs
Required Mission Fuel Weight	567 lbs
Empty Weight	2176 lbs

Table 19. Case Study II: Breakdown of the Required Mission Fuel Weight

Mission Segment		Required Fuel Weight (lbs)
1 ~ 2	Warmup	1.08
2 ~ 3	Takeoff	2.4
3 ~ 4	Climb	11.167
4 ~ 5	Cruise	478.35
5 ~ 6	Loiter	74.29
6 ~ 7	Descend	-
7 ~ 8	Landing	-

Figure 73 is mainly about the changes of different weights, including takeoff gross weight, empty weight, required aviation fuel weight, and required hydrogen fuel weight, corresponding to the change of DFP. Figure 74 shows the comparison of the weights of the two engine systems and the corresponding fuel weights under different DFP values. The results in these two figures are reasonable. For small DFPs, the weight of the fuel cell system accounts for as large as nearly 50% of the takeoff gross weight. As DFP increases, both the fuel cell system weight and the weight of hydrogen fuel decrease; however, the weights of the piston engine and required aviation fuel increase.

Figure 75 is an area chart on weight study of the hybrid propulsion system. It provides another view of the weight change corresponding to the change of DFP. In this graph, the areas in between two adjacent lines shows the changes of weights, particularly the weights of hydrogen fuel, aviation fuel, piston engine, fuel cell system, and hydrogen storage.

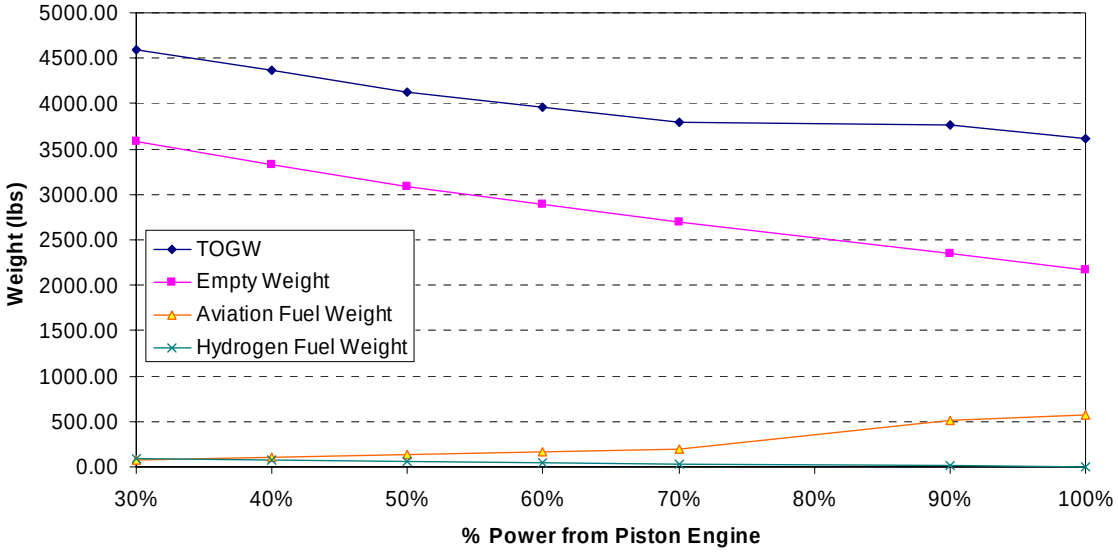

Figure 73. Case Study II: Study Result of Weights

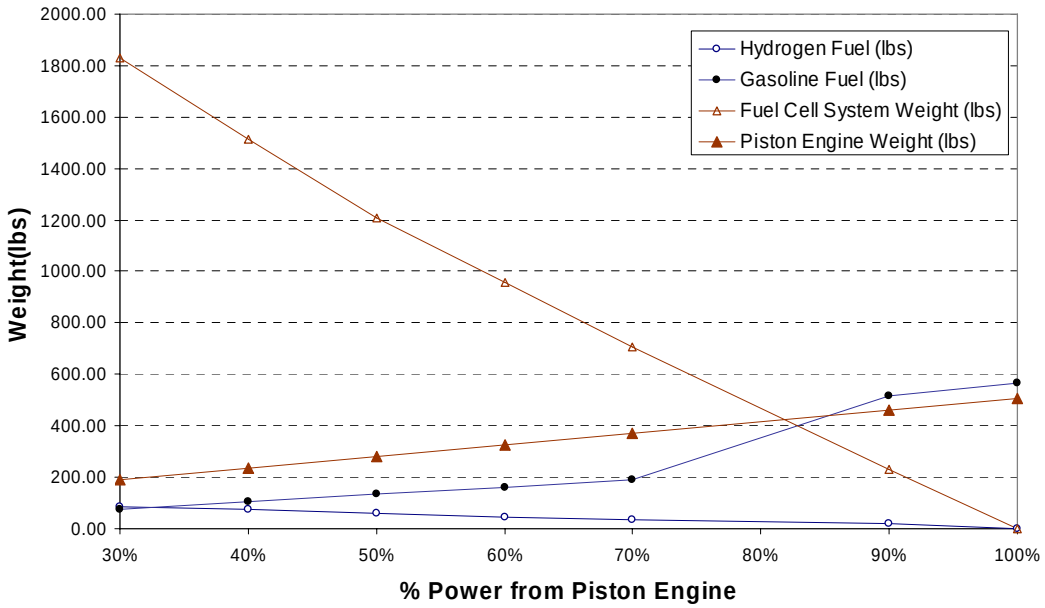

Figure 74. Case Study II: Weight Study Result of the Hybrid Propulsion System (1)

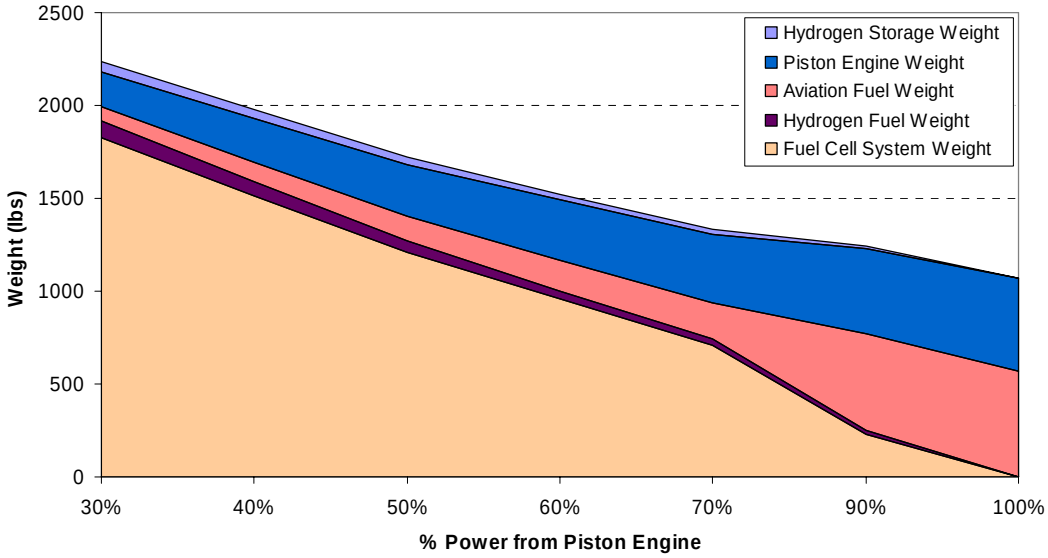

Figure 75. Case Study II: Weight Study Result of the Hybrid Prolusion System (2)

Figure 76 is a bar chart showing the breakdown of required mission fuel weights including hydrogen fuel and aviation fuel into different mission segments for the case of DFP=50%. The figure presents an interesting phenomenon - when the power is distributed equally to the two engine systems, the required weight of hydrogen fuel is always much less than the required weight of aviation fuel. It seems that hydrogen fuel is a good choice compared with aviation fuel. However, considering the weight of the fuel cell system as shown in Figure 74, the attractiveness of lightweight hydrogen fuel may not be quite obvious. So in order to get the optimum takeoff gross weight, one of the tradeoff studies should be the distribution of the power between the two engine systems.

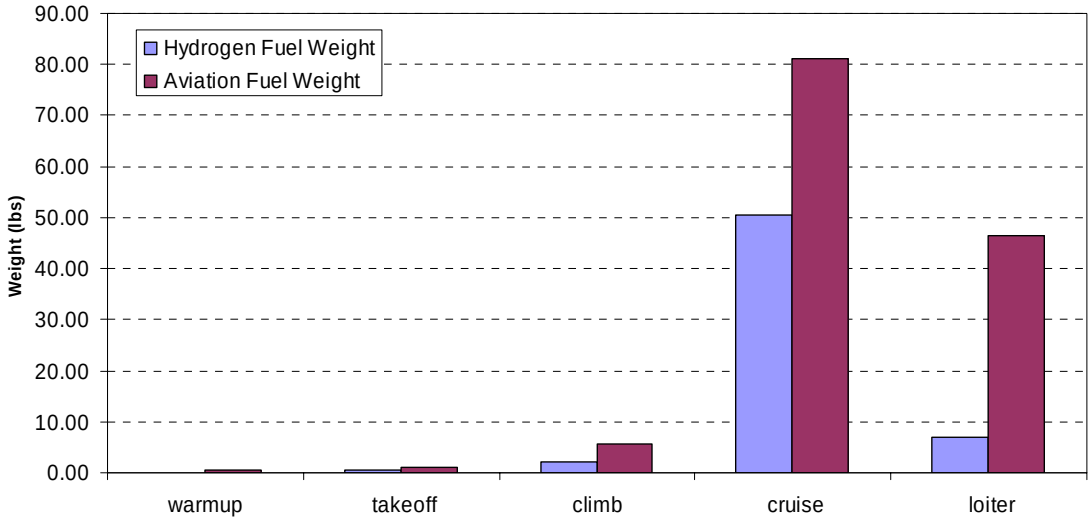

Figure 76. Case Study II: Mission Fuel Weights Decomposition for DFP=50%

From these results, the conclusion can be made that the PEMFC technology is not beneficial to the hybrid propulsion system because of the heavy weight of the fuel cell system utilizing this technology. At the same time, this case study does show the ability provided by the proposed object-oriented conceptual design framework to facilitate the design and analysis of unconventional aircraft concepts.

7.4 Observations and Discussions

There is no doubt that the traditional conceptual design environments being used in practice cannot fully meet the emerging requirements of aircraft conceptual design, and the next generation aircraft conceptual design environment concept should be formulated and examined. These two case studies provide the author some insights on 1) how well

the proposed object-oriented distributed environment and the data management system solve the research questions, which are the major problems that need to be resolved regarding the traditional aircraft conceptual design environments being used in practice; and 2) what the advantages and disadvantages of the prototype integrated design environment built based upon these proposed solutions are compared to traditional aircraft conceptual design environments. In this section, some observations made in the case studies will be discussed from the viewpoints of domain flexibility, analysis scalability, and interoperability. For aircraft conceptual design environments, these three most important desired traits are mutually related to each other. The improvement made in one aspect will enhance the others. For example, improved interoperability of the analysis programs in the design environment will make it easier to promote domain flexibility and analysis scalability. However, in order to make the points clear, they will be discussed one-by-one in the following sections.

7.4.1 Domain Flexibility

The proposed object-oriented design framework makes it possible to achieve domain flexibility in the integrated design environments built for the two case studies. For Case Study I, the algorithm used to design the notional conventional fighter falls into the category of dissipative fuel based aircrafts. As manifested in Case Study I, the integrated design environments built based upon the proposed framework handle it very well - the same as traditional aircraft conceptual design programs. In Case Study II, which is different from Case Study I, the algorithm adopted to design the Cessna Hybrid is for designing aircrafts that consume both dissipative and non-dissipative fuels. None of the

traditional aircraft conceptual design environments can be used to design the Cessna Hybrid due to the facts that the built in algorithms of these design programs are not suitable for designing such an aircraft, and the amount of work to change the design algorithms is overwhelming. However, as concluded from Case Study II, the new integrated design environments are fully capable to meet the requirements of designing unconventional aircraft.

In the proposed object-oriented distributed design framework, domain flexibility is achieved because the algorithms are treated as something variable, which depends on the design requirements. In a sense, it is to say that the design algorithm is an input that needs to be fed into the design environments, similar to a regular design variable. Therefore, unlike traditional aircraft conceptual design programs, whose built-in algorithms are treated as something fixed, the proposed objected-oriented approach gives designers more flexibility. As mentioned previously, domain flexibility and analysis scalability are mutually related to each other. Without analysis scalability, achieving domain flexibility is impossible. By using the proposed object-oriented approach, the building blocks of an aircraft conceptual design environment are modeled as standalone objects, designers have the flexibility to choose which analysis programs to use for different design algorithms. This is in contrast to the traditional design programs, with which designers have no alternatives but the analysis functionality built into them.

Besides analysis programs, design algorithms are also implemented in the design and analysis processes. When using the traditional aircraft conceptual design programs, designers cannot change the design and analysis processes. They are hard coded into the design programs together with the design algorithms. They are fixed. What designers

need to provide and can change are the values of some variables. However, as shown in the two case studies, designers can not only set the values for the variables but also change the design and analysis processes with the proposed object-oriented approach. This is a double-edged sword. On one hand, designers have more flexibility. On the other hand, compared to traditional design programs, trust in the design results partly relies on how well designers model these processes. Depending on how readers look at this, it seems to be a disadvantage of the proposed framework at this research stage because a wrongly modeled process might also jeopardize the efficiency and robustness of a design environment, especially when it becomes very complex. The author thinks that it is a tradeoff that has to be made. If we look at it the other way, unlike the black box approach adopted by the traditional conceptual design program, this helps designers to understand the design problem more completely, and gives them more freedom and space to develop better design concepts.

7.4.2 Analysis Scalability

These analysis programs integrated in the design environments constitute two first order analysis and modeling environments for the design and analysis of the conventional notional aircraft in Case Study I and the Cessna Hybrid in Case Study II. In these integrated design environments, each of the analysis programs is modeled as an object that is independent from other analysis programs, which are wrapped as objects as well. It communicates with other related analysis programs but it does not need to know how the analysis logics are implemented in other analysis programs, and its own analysis logic is kept within the object itself. As introduced previously, this is termed encapsulation.

Encapsulation helps to keep the data and implementation within the objects – localizing the knowledge. This characteristic of the object-oriented design approach enables us to achieve analysis scalability in the integrated design environments. Designers can add, remove, or change an analysis object as needed without affecting other objects. The capability to perform variable fidelity analyses is a desired trait. For example, for the design of the Cessna Hybrid in Case Study II, if designers want to perform an extended study on the hybrid propulsion system, the current first order *Matlab* analysis programs integrated into the design environment can be replaced with an higher order modeling and analysis program(s). Nothing needs to be changed for other analysis programs but the wrappers that mainly handle inputs and outputs. However, without rewriting and even restructuring the whole programs, it won't be an easy task for most of the traditional aircraft conceptual design programs.

The support of the central data management system enhances the analysis scalability of the integrated design environments. Without the support of the database, the changes of one analysis object will affect all the objects that exchange information with it. Changes to the wrappers may be required, and the communication channel needs to be rebuilt. For example, all the seven mission analysis objects in Case Study I rely on the aerodynamic analysis object for aerodynamic information, such as, drag coefficient and reference area. If the aerodynamic analysis object needs to be replaced using another one in order to meet new requirements, changes should be made to the wrappers of all the mission analysis objects, and the communication mechanism between the new analysis object and the seven mission analysis objects needs to be established again one-by-one. However, the utilization of the central data management system will simplify this process.

For the same scenario described above, the seven mission analysis objects communicate with the database for the aerodynamic information they need. The only wrappers to be written are those that enable the communication between the new aerodynamic analysis object and the database. Changes don't need to be made to the wrappers of the seven mission analysis programs.

Speaking of mission analysis, how the mission is being modeled also contributes to analysis scalability and domain flexibility of the integrated conceptual design environments. In the proposed framework, the mission is decomposed into atomic mission segments and each of these segments is modeled as an object. It can be seen from the two case studies that designers can construct a variety of mission profiles for different design concepts or different design algorithms in the integrated design environments. As in the above discussion, the central data management system makes it easier for changing a mission segment (a mission analysis object) or the entire mission.

7.4.3 Interoperability

Improving the interoperability of the NextADE is the major focus of this research. Comparing the integrated design environments with and without database support built for the two case studies and the traditional conceptual design programs, the conclusion can be made that the use of the centralized data management system will make the NextADE perform better in terms of interoperability and, in turn, flexibility and scalability. This does not mean that the interoperability of the latter is poor. Through translating the definition, modeling of domains, and parsing data using text-based

input/output files, traditional design programs can interoperate with other software systems, although not quite efficiently when the system becomes large and complicated.

How to assess, measure, and quantify interoperability is still a topic that is being researched. A widely recognized achievement under this topic is the Levels of Information System Interoperability (LISI), which is a reference model and process for assessing information systems' interoperability, developed by the DoD C4ISR Working Group [128]. The LISI model depicts five levels of interoperability maturity, as illustrated in Table 20 and Table 21. According to the LISI model, traditional aircraft conceptual design programs would have an interoperability maturity level of Level 1 (the Connected level), while the interoperability maturity level of the NextADE, which is supported by a centralized data management system, would be at Level 3 (the Domain level) featured with domain models, shared databases, integrated computing environments, and sophisticated collaboration using networks.

Table 20. LISI Five Levels of Interoperability Maturity [124]

Level	Information Exchange
4. Enterprise Interactive manipulation Shared data and applications	Distributed global info. and apps. Simultaneous interactions w/ complex data advanced collaboration e.g. interactive COP update Event-triggered global database update
3. Domain Shared data Separate applications	Shared databases sophisticated collaboration e.g. common operational picture
2. Functional Minimal common functions Separate data and application	Heterogeneous: product exchange Basic collaboration Group Collaboration e.g. Exchange of annotated imagery, maps w/ overlays
1. Connected Electronic connection Separate data and application	Homogeneous: product exchange e.g. FM voice, transfers, messages, emails
0. Isolated Non-connected	Manual Gateway e.g. diskette, tape, hard copy exchange

Table 21. LISI Maturity Levels vs. Interoperability Attributes

(Adapted from [128, 129])

Level	Procedure	Application	Infrastructure	Data	Computing Environment
4. Enterprise	Enterprise Level	Interactive	Multiple Dimensional Topologies	Enterprise Model	Universal
3. Domain	Domain Level	Groupware	World-Wide Networks	Domain Model	Integrated
2. Functional	Program Level	Desktop Automation	Local Networks	Program Model	Distributed
1. Connected	Local/Site	Standard System Driver	Simple Connection	Local	Peer-to-peer
0. Isolated	Access Control	N/A	Independent	Private	Manual

7.4.4 Some Other Observations

Some other observations were also made while conducting the two case studies. Summarized in Table 22 are some facts of the two case studies. The computing times in the table are average numbers in minutes for a PC with Intel Pentium 4 CPU 1500MHz processor and 392MB RAM. The other numbers, including the number of analysis programs integrated, the number of *ModelCenter* wrappers used for integration, and the amount of variable stored in the database are for illustration purpose only. The values of these numbers really depend on which analysis programs and integration tools are used. Also, for different design problems, the numbers of design variables will vary accordingly.

- As the comparison made in Table 22 in terms of computing time, for Case Study I, Using ACSYNT, it takes the least amount of time to get the final results. And for both of the two case studies, it appears that the computing

Table 22. Some Facts of the Case Studies

	Tradition Design Program (ASCYNT)	New Integrated Design Environments			
		W/O Database Support		W/ Database Support	
		Case Study I	Case Study II	Case Study I	Case Study II
Computing Time	Fast (<10 mins)	Slow (20 ~ 30 mins)	Slow (20 ~ 30 mins)	Slowest (30 ~ 40 mins)	Slowest (30 ~ 40 mins)
Number of Analysis Programs Integrated	-	11	14	11	14
Numbers of ModelCenter Wrappers Used for Integration	-	11	14	35	38
Number of Variables Stored in the Database	-	-	-	261	279

time of finishing a design process in the integrated design environment supported by a database is several minutes longer than finishing the same design process in the environment without the support of such a database. Most of the traditional aircraft conceptual design programs are self-contained "black boxes", which only need to run on a single computer. It is an advantage when it comes to computing time. Contrary to this, the openness of the proposed design framework, which usually needs the involvement of several software and hardware systems, will slow down the computation as shown in the table.

Additionally, the integrated design environments are constructed using a general-purpose integration tool, *ModelCenter*, and it is certain that the performance of *ModelCenter* affects the computing time. However, it is hard to measure how much effect the integration tool has at this research stage. From Table 22 and Appendix J, it can be seen that more *ModelCenter* wrappers – twenty-four more for both Case Study I and Case Study II - are needed in order to integrate the database management system into the design environment. The execution of these wrappers accounts for increased computing time. Besides the performance of the integration tool, the performance of different hardware is a factor that should be considered. For example, the configuration of the computer used to conduct the experiments, and *MySQL* and *Matlab* used for Case Study II.

Questions might be asked, such as "*does modeling the design data take time as well, and will this slow down the aircraft conceptual design process?*"

To the first question, the answer is yes. When we first build such an integrated conceptual design environment, we need to spend time on carefully modeling the design data and building the data management system. However, this is a one-time effort and belongs to the stage of designing and developing the integrated design environment, not an aircraft. For aircraft designers, in an integrated design environment supported by a central data management system, as examined in the two case studies, the design process will be simplified, and thus, they will spend much less time on modeling the design process. This is why it is important to have a data model that is general enough for different design concepts while still able to capture more detailed information. The proposed project-based object-oriented data model for aircraft conceptual design described in Chapter 5 fulfills this purpose. It is possible that a new aircraft or updates to existing analysis tools integrated into the design environment necessitate modifying the data model and updating the framework. The author believes this will be rare, but whether this is so in actual practice remains to be seen.

- Another observation made is that we can build the integrated design environments using general-purpose integration software packages as has been done for the two case studies; however, this may not always be the best strategy. In the design environment, if a customized built-in database is needed, and a GUI should be designed and developed, performance would likely be improved, especially computing. Such requirements may call for a standalone, more productive, dedicated, specialized design environment.

7.4.5 Summary

In summary, a comparison between the NextADE and traditional aircraft conceptual design environments is made according to the desired traits listed in Table 5. Such comparison is very helpful for understanding the advantages and disadvantages of the proposed framework and the data management approach. The comparison is made on a qualitative base. The author feels that at this stage it is difficult to make a quantitative comparison due to the fact that a dedicated standalone NextADE has not been developed yet. The design environments constructed for the experiments in this chapter are integrated using a general-purpose integration software package. Making a quantitative comparison based upon the performance of such environments is very hard if not biased. However, in the future, during the development stage and the maintenance stage after a standalone dedicated NextADE has been developed, studies can and should be conducted systematically using software quality engineering metrics and measures [130]. For example, we can track the mean time to failure (MTTF) in order to quantitatively evaluate the reliability and robustness of the software system. The concept of MTTF of a software system is very similar to the MTTF we use to measure the safety of a safety-critical system, such as aircrafts and weapons, which is defined as the average time between unexpected events.

The result of the comparison is summarized in Table 23. For convenience, the definition of each desired trait is also listed in the table. The comparison results of "Better", "Good", and "Poor" are given based upon the relative performance of the three options only. There are no other known benchmark conceptual design programs. As has been explained, traditional aircraft conceptual design programs are well trusted in their

Table 23. Comparison between the NextADE and Traditional Conceptual Design Programs

Desired Trait	Definition	New Integrated Design Environments		Traditional Conceptual Design Programs
		W/ Database Support	W/O Database Support	
Efficiency	The ability to do a task within certain constraints, such as time and memory space	Good	Good	Better
Robustness	The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions [2]	Good	Good	Better
Scalability	The ease with which a system or component can be modified to fit the problem area	Better	Good	Poor
Flexibility	The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [2]	Better	Good	Poor
Extensibility	The degree to which an application or component is able to be enhanced in the future to meet changing requirements or goals	Better	Good	Poor
Interoperability	The ability of two or more systems or components to exchange information and to use the information that has been exchanged [2]	Better	Poor	Poor
Reusability	The characteristics that codes written for one application can be reused in different applications	Better	Better	Poor

specific domains. However, due to the availability of computing technologies when they were developed and the advancement of aircraft conceptual design, they present limitations to meet the current and future requirements in practice – for example, to perform variable fidelity analyses and design unconventional aircrafts. The major obstacles are the lack of domain or algorithm flexibility, analysis scalability, and interoperability. As has been demonstrated and explained with the experiments conducted in the two case studies and the discussion made in the previous several sections, the proposed object-oriented distributed framework makes it possible to achieve domain flexibility and analysis scalability. Furthermore, the central database management system improves the interoperability of such design environments. Therefore, the integrated design environment with the support of the central data management system performs better in these three aspects; the integrated design environment without the support of the data management system falls in between; and the traditional aircraft conceptual design program are not as good as the other two.

However, there is no free lunch - maximizing some qualities may cause others to decrease. Such capabilities of the NextADE might be achieved with the sacrifice of some other qualities, such as efficiency and robustness. To mitigate this conflict, a tradeoff needs to be made. In terms of model execution efficiency, which is the extent to which software uses minimum hardware resources to perform its functions, traditional aircraft conceptual design programs are better compared to the integrated design environments due to the fact that the proposed object-oriented distributed design framework might need more hardware capabilities, for example more memory space and those hardware on the Internet. Related to model execution efficiency, the robustness of the integrated design

environments with and without the support of the database might not be as good as the traditional conceptual design programs, not only because that it needs more hardware capabilities but also because that the software system itself is more complicated, especially for a design environment support by a data management system.

It has been discussed that the lack of openness and the procedural approach, which traditional aircraft conceptual design programs adopt, make it difficult for us to scale, extend, and reuse them. However, in addition to flexibility, scalability, and interoperability, the object-oriented design of the proposed framework also provides us other such -ilities including extensibility and reusability. These are the advantages of the object-oriented approach over the traditional procedural approach when being used for the design and development of a large complex software system [9]. Therefore, regarding extensibility and reusability, the integrated design environments will perform much better than traditional aircraft conceptual design programs.

It can been seen from the two case studies and the discussions made in the previous several sections that for simple conventional aircraft conceptual design problems, traditional aircraft conceptual design programs are better choices for better efficiency and robustness, if the built-in algorithms and the analysis functionality of them are suitable for the design problems. However, for complex conceptual design problems, for example, unconventional aircrafts and variable fidelity analyses, with improved domain flexibility, analysis scalability, and interoperability, the proposed integrated objected-oriented distributed approach will benefit designers the most.

CHAPTER 8

CLOSING REMARKS AND RECOMENDATIONS

As the closure of this dissertation is reached, the research conducted during the course of this dissertation is summarized in this chapter. Future works are recommended at the end.

The objective of the research is to find an effective data management approach for the next generation aircraft conceptual design environment, which should be a distributed object-oriented design environment. Due to the fact that we do not have such a promising design environment yet, a proposed framework of this new design environment is formulated and tested first. Then, according to this framework, taking into account the characteristics of aircraft conceptual design data, a robust, extensible object-oriented data model is proposed. Based upon this data model, a prototype of the data management system, which should be one of the fundamental building blocks of the next generation aircraft conceptual design environment, is developed utilizing the state-of-the-art information technologies.

To demonstrate the efficacy of the proposed framework, a data model, a prototype data management system, and the next generation aircraft conceptual design environment (NextADE) are initially implemented by integrating the data management system with other building blocks using a general-purpose integration software system. Two case studies, a simplified conceptual design of a notional conventional aircraft and a simplified conceptual design of an unconventional aircraft, were conducted in the

integrated design environments for experimentation. As a result following the experiments, the proposed framework and the data model are determined to be feasible solutions to the research problems.

8.1 Answers to Research Questions

Two research questions are posed in Chapter 2. Each of them has been answered during the course of this dissertation. Following is a summary of the answers to the two research questions:

- Question #1: *How should we draw the "blueprint" of the NextADE, i.e. considering the building blocks constituting an aircraft conceptual design environment and the desired traits, particularly interoperability, flexibility, scalability, and reusability, how should the framework of the NextADE be formulated?*

  This question is answered by finishing the three tasks of 1) decomposing the ideal design environment and developing resulting building block concepts, 2) prototyping and testing the decomposed building blocks, and 3) integrating these building blocks to construct an initial prototype design environment. The ideal design environment is decomposed object-orientedly into components of mission analyses and contributing disciplinary analyses. At the sublevel, mission analyses are decomposed into elementary

mission segment objects based upon the mission profile. For most disciplinary analysis components, legacy programs can be reused using wrappers. The mission analysis component was completely reformulated and then tested in an integrated design environment.

In the proposed design framework, domain flexibility and analysis scalability are achieved. It was demonstrated that unconventional concept vehicles could be designed in the design environment. Flexibility is provided for designers to perform variable fidelity disciplinary analyses. Thus, Hypothesis I, which states that *"the object-oriented design and analysis approach is an effective approach that can be employed for the design and development of the NextADE",* is demonstrated.

However, there is a note of caution in that the validity of the results of this objected-oriented design environment depends heavily on the correctness of the construction of the design process. Effective data management can help to relieve this problem and the answer to the next research question shows how this can be achieved.

- Question #2: *How can design data be managed effectively in the NextADE, instead of using the traditional file processing approach?*

This research question is answered after finishing the tasks related to it. According to the proposed distributed object-oriented framework of the design environment, in order to manage design data effectively, another important building block, a central integrated data management system is

added to the proposed framework. An object-oriented data model is proposed for this data management system. The difficult problem for modeling aircraft conceptual design data in such a design environment is that the model should be general enough to accommodate different vehicle concepts. This model overcomes the shortcomings of how people traditionally model aircraft conceptual design data. It is very robust and extensible. This is achieved mainly through separating analyses or geometry components related attributes from their corresponding object classes, and modeling them as standalone objects. Doing so enables the model to capture enough details of design data without losing generality.

In the implementation fro verification purposes, a prototype of the data management system is developed. Certain functionalities for query and data management are provided by this data management system. The use of a combination of object-oriented programming, relational database management technologies, and XML technologies are explored for such development by taking advantage of what each of these technologies provides us.

Then, for the experiments of two case studies, using a general-purpose commercial integration software package, integrated aircraft conceptual design environments are prototyped based on the above concepts by integrating the data management system with other building blocks. In order to demonstrate the flexibility, scalability, and interoperability achieved in such environments, the two case studies are chosen to be very different. One of them is based upon a simplified conceptual design of a notional conventional

aircraft. The other one is conducted for a simplified conceptual design of an unconventional aircraft.

The experiments show that the proposed approach can manage design data effectively in the distributed object-oriented design environment. The use of the centralized data management system improves data sharing among heterogeneous analyses programs, simplifies the design process, and encourages design analyses to be conducted concurrently when possible. Also by isolating the other building blocks from each other, the data management system is also critical for achieving domain flexibility and analysis scalability. This supports Hypothesis II, which claims that *"effective data management is the key to improving the communication among the heterogeneous design and analysis computer programs involved in aircraft conceptual design and thus promoting the efficiency of the overall design process, given that the design methodology (such as sizing algorithms) and the disciplinary analysis computer programs (such as programs of optimization, aerodynamic analysis, propulsion, structure analysis, etc.) needed by a design are available and accessible"*.

8.2 Summary of Contributions

The major achievements and contributions of the research in this dissertation include:

- A object-oriented distributed aircraft conceptual design framework is formulated for the next generation aircraft conceptual design environment in order to achieve the desired traits, particularly flexibility, scalability, reusability.
    - Through the utilization of the object-oriented design and analysis approach, the proposed framework overcomes the limitations, mainly domain flexibility and analysis scalability, imposed by the traditional aircraft conceptual design programs to meet the current and future design requirements, for example performing variable fidelity disciplinary analyses and unconventional aircraft conceptual design.
    - An object-oriented mission analysis approach is also formulated during the research. As a practical contribution, an object-oriented mission analysis component is completely reformulated and developed.

- Another obstacle, the interoperability problem, in the above object-oriented distributed next generation aircraft conceptual design environment is addressed. A data management approach is proposed for solving this problem. The proposed data management system is also important for improving domain flexibly and analysis scalability of such design environments. Providing a feasible solution to this problem is the major achievement of this research.

- A robust, extensible object-oriented data model for aircraft conceptual design data is proposed. One of the major features that distinguish the data model from how others model aircraft conceptual design data is the separation of analysis or geometry component-related attributes from their corresponding object classes. Instead, these attributes are being modeled as separated but related objects, which enable the data model to capture enough details of design data without losing generality.
- Based upon this data model, a prototype aircraft conceptual design data management system is developed. This is a practical contribution.

- An initial integration of the NextADE is conducted using a general-purpose integration software package. The implementation result shows that we can build the integrated design environments using general-purpose integration software packages. However, it may not be the best strategy depending on the objective of a specific environment. For example, a standalone, more productive, dedicated, specialized design environment will likely provide better execution performance.

- Along with solving the research problems and testing the proposed solutions via implementation, the use of state of the art computing technologies on the design and development of an aircraft conceptual design environment is explored.

- The problem characterization (including identifying key metrics for such design environments), the test cases, and the solution approach provide a baseline contribution which future researchers may leverage and compare against.

Recently, with the rapid development of technologies, researchers came out with the concept of system of systems. Except the difference of domains, the design of an aircraft, which itself can be considered as a system of systems, and the design of other complex system of systems shares some similarities [131]. Therefore, progress made and lessons learned from this research work may prove to be useful for the design of other complex systems of systems.

8.3 Recommendations

During the course of the research in this dissertation, several areas of future work have been identified. They are recommended as following:

- Systematic investigation on the usage of different database management approaches:

  In this dissertation, object-oriented technologies are used for design, analysis, and implementation programming; XML technologies are used for data exchange; and a relational database is used as the supporting database. In the future, as object-oriented database management and XML database management technologies

become relatively mature, a systematic investigation should be conducted on using each of these major databases as the supporting database. The purpose of this investigation is to find the most appropriate one or the most appropriate combination of database management approaches.

- Process modeling and management:

  As mentioned previously, in the proposed distributed object-oriented design environment, trust on the result lies more on building the design process correctly, which includes verifying the resulting process in some may. Therefore, besides an effective data management approach, we are also in need of an effective process management approach. A near term research plan should be scheduled focused on solving this problem.

- Disciplinary analyses:

  This is not the focus of the research in this dissertation. However, it has been realized that the proposed distributed object-oriented aircraft conceptual design framework may impose some requirements on disciplinary analyses, especially optimization and interactive geometry design integration. Research should be conducted to find out and then cover such problems.

- Development of a specialized, dedicated, object-oriented aircraft conceptual design environment:

  As has been recognized, using general-purpose integration tool as the integration vehicle to build this design environment may not be the best strategy depending on the objective of a specific design environment. A specialized, dedicated,

distributed object-oriented aircraft conceptual design environment will likely provide better execution performance. Such a design environment should be developed. Also, once it is developed, if possible, we can quantitatively measure the quality of the new environment in terms of efficiency, robustness, flexibility, scalability, interoperability, and reusability in a meaningful way.

- Investigation of general system of systems capabilities:

  It should be investigated if general system of systems capabilities (including SysML) can provide effective foundations for the NextADE. For example, the SysML metamodel comes many of the same concepts as the proposed NextADE data model.

- Integration of the conceptual data management system with other data management systems used in later design stages:

  At this stage, the research focuses on finding the right solution to manage aircraft conceptual design data effectively due to its special characteristics. In light of AEEs, for managing the information of the entire lifecycle of an aircraft, time should be spent on how to integrate aircraft conceptual design data management with later design stages, especially preliminary and detail design stages.

- From data to knowledge:

  Technologies for knowledge management should be investigated for aircraft design. Knowledge management deals with the processing of meta-data rather than just data [132] and with the representation of relations among objects (e.g. symbolic equations). In this aspect, XML deserves more attention; and

technologies such as ontology, which is a representation vocabulary specialized to some domain or subject matter, and Resource Description Framework (RDF) [133, 134], which is an enabling technology for semantic modeling, provide good starting points.

APPENDIX A. Implementation of the Mission Analysis Component

As an example demonstrating how the mission segments objects are implemented, the source code, the input and output files for the cruise segment analysis are given in this appendix.

A.1 Source Code

Cruise Segment Analysis:

```
/**
 * <p>Title: Cruise.java</p>
 * <p>Description: Calculate the weight and weight fraction after
 *                 constant speed/altitude cruise
 *                 based on energy balance.</p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: ASDL</p>
 * @author Zhijie LU
 * @version 1.0
 */

import java.io.*;
import java.util.*;

public class Cruise extends mission {
  private double mach, range, altitude;
  private int IPS;
  private String inputFileName = "cruise.in";
  private String outputFileName = "cruise.out";

  public Cruise(){}

  public Cruise(String inputFileName, String outputFileName) {
    this.inputFileName = inputFileName;
    this.outputFileName = outputFileName;
  }

  public void run(){
    System.out.println("Cruise:");
    Properties prop=null;
    try{
      FileInputStream fis=new FileInputStream(inputFileName);
      prop=new Properties();
      prop.load(fis);
      String strIW=prop.getProperty("initialWeight", "0");
      String strMach=prop.getProperty("mach", "0");
      String strRange=prop.getProperty("range", "0");
      String strAltitude=prop.getProperty("altitude", "0");
      String strIPS=prop.getProperty("IPS","0");

      initialWeight=Double.valueOf(strIW).doubleValue();
      mach=Double.valueOf(strMach).doubleValue();
```

```
    range=Double.valueOf(strRange).doubleValue();
    altitude=Double.valueOf(strAltitude).doubleValue();
    IPS=Integer.valueOf(strIPS).intValue();
  }catch(Exception e){
    e.printStackTrace();
    System.exit(1);
  }
  this.calcFinalWeight();
  double result=this.getFinalWeight();
  double ratio=this.getWeigthRatio();

  //OUTPUT
  System.out.println("Final Weight="+result);
  System.out.println("Weight Ratio="+ratio);
  try{
    FileOutputStream fos=new FileOutputStream(outputFileName);
    prop.setProperty("finalWeight",   Double.toString(result));
    prop.setProperty("weightRatio", Double.toString(ratio));
    prop.store(fos, "Cruise Segment: ");
  }catch(Exception e){
    e.printStackTrace();
    System.exit(1);
  }
}

public void setInputFileName(String inputFile){
  this.inputFileName = new String(inputFile);
}

public void setOutputFileName(String outputFile){
  this.outputFileName = new String(outputFile);
}

public void setMach(double mach){
  try{
    Properties prop=new Properties();
    FileInputStream fis=new FileInputStream(inputFileName);
    prop.load(fis);
    FileOutputStream fos=new FileOutputStream(inputFileName);
    prop.setProperty("mach",   Double.toString(mach));
    prop.store(fos,"");
  }catch(Exception e){
    e.printStackTrace();
    System.exit(1);
  }
}

public void setRange(double mach){
  try{
    Properties prop=new Properties();
    FileInputStream fis=new FileInputStream(inputFileName);
    prop.load(fis);
    FileOutputStream fos=new FileOutputStream(inputFileName);
    prop.setProperty("range",  Double.toString(range));
    prop.store(fos,"");
  }catch(Exception e){
    e.printStackTrace();
    System.exit(1);
  }
}

public double getMach(){
  return mach;
```

```
  }

  public double getSpeed(){
    return mach * Atmosphere.getSoundSpeed(altitude);
  }

  public double getRange(){
    return range;
  }

  public double calcCL(double lift, double rho, double speed, double refArea){
    double CL = 2 * lift/(rho*speed*speed*refArea);
    return CL;
  }

  public void setInitialWeight(double initialWeight){
    try{
      Properties prop=new Properties();
      FileInputStream fis=new FileInputStream(inputFileName);
      prop.load(fis);
      FileOutputStream fos=new FileOutputStream(inputFileName);
      prop.setProperty("initialWeight", Double.toString(initialWeight));
      prop.store(fos,"");
    }catch(Exception e){
      e.printStackTrace();
      System.exit(1);
    }
  }

  public void setAltitude(double altitude){
    try{
      Properties prop=new Properties();
      FileInputStream fis=new FileInputStream(inputFileName);
      prop.load(fis);
      FileOutputStream fos=new FileOutputStream(inputFileName);
      prop.setProperty("altitude", Double.toString(altitude));
      prop.store(fos,"");
    }catch(Exception e){
      e.printStackTrace();
      System.exit(1);
    }
  }

  public void calcFinalWeight() {
    double rho = Atmosphere.getDensity(altitude);
    double speed=this.getSpeed();
    double refArea = Aero.getRefArea();
    double lift = initialWeight;
    double CL = this.calcCL(lift, rho, speed, refArea);
    double CD = Aero.getCD(altitude, mach, CL);// call Aero
    double drag = 0.5*rho*speed*speed*refArea*CD;
    double thrust = drag;  // required thrust
    double TSFC= Prop.getTSFC(altitude, mach, thrust, IPS)/3600;  // 1/sec

    finalWeight=initialWeight - thrust * TSFC * range/speed;
  }

  public static void main(String args[]){
    Cruise cruise = new Cruise(args[0], args[1]);
    cruise.run();
  }
}
```

A.2 Input File

The input file, named *cruise.in*, for the cruise segment analysis:

```
#
#Mon May 17 22:44:02 EDT 2004
altitude=25000.0
initialWeight=26479.0029593659
range=341783.0
IPS=4
mach=0.95
```

A.3 Output File

The output file, named cruise.out, of the cruise segment analysis:

```
#Cruise Segment:
#Tue Aug 30 21:30:46 EDT 2005
altitude=25000.0
initialWeight=26479.0029593659
range=341783.0
IPS=4
mach=0.95
weightRatio=0.9731606835362161
finalWeight=25768.324619294006
```

APPENDIX B. Hibernate and XDoclet

Hibernate [107] is a Professional Open Source project. It was originated by Gavin King in late 2001 in order to find a solution for the object relational paradigm mismatch, which refers to the problem caused by the difference between the tabular representation of data in a relational system and the networks of objects used in object-oriented applications. In late 2003 the Hibernate team joined JBoss Inc., which provides commercial support and training. Hibernate provides high performance object relational persistence and query service for Java. It enables users to develop persistent classes that are mapped to database tables according to the common Java idioms such as association, inheritance, composition, polymorphism, and the Java collections framework. It also allows users to express queries in SQL and its own portable SQL extension (HQL). Another important feature of Hibernate is that the persistent class can be used in an execution context without a container. In contrast, some other implementation of persistent layers is implemented as a framework or container that imposes special design constraints on the classes to be persisted. Typically, it is difficult to run or test applications without the framework environment. The objects can only work in the framework/container and they cannot be passed from the container to the outside. Developers have to copy the objects from one tier to another tier, for example, from the server tier to the client tier.

Hibernate provides five core programming interfaces: the *Session* interface, the *SessionFactory* interface, the *Configuration* interface, the *Transaction* interface, and the *Query* and *Criteria* interface. Shown in the Figure 77 is the runtime architecture of

Hibernate [135]. The collection of *Persistent objects* can be ordinary JavaBeans. They are single threaded and short lived. They contain persistent state and business function. A *Session* object is a single-threaded short-lived object representing a conversation between the application and the database. The *SessionFactory* is a factory for *Session*. It is a thread safe cache of compiled mappings for a single database. *ConnectionProvider* is an optional factory for JDBC (Java Database Connectivity) connections. A *Transaction* object is a single-threaded, short-lived object which abstracts application code from the underlying transaction implementation. It is also optional. The *TrasactionFactory* is an optional factory of the *Transaction* objects.

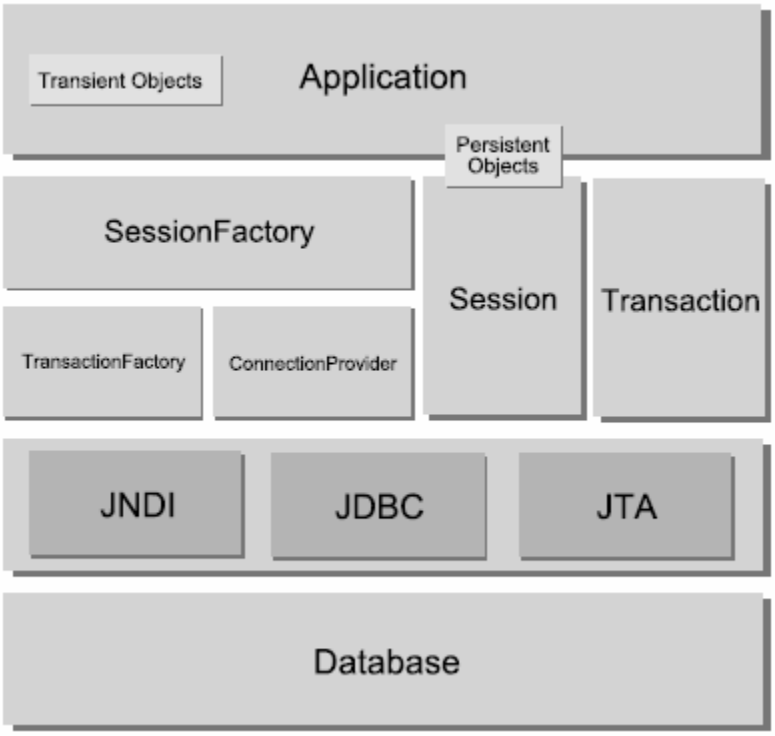

Figure 77. The Hibernate Runtime Architecture [135]

In a nutshell, the following describes how Hibernate works. First of all, Hibernate needs mapping files (hbm.xml files) to persist the data objects into database tables. In these mapping files, we need to be able to tell what attributes are to be persisted and what the relationships are. We need to provide the mapping file for each data object. These mapping files are loaded into the Hibernate *Configuration*. From the *Configuration*, we can create a *sessionFactory*. The *sessionFactory* enables us to create *Session* objects. Through these *Session* objects, the application can communicate with the database.

Xdoclet [108] is an open source code generation engine that enables attribute-oriented programming for Java. It uses the information provided in the source code and its custom JavaDoc @tags to generate external resource files, such as XML descriptors, to support the main Java classes. However, currently XDoclet can only be used as part of the build process utilizing Jakarta Ant [136].

XDoclet has a set of modules to generate different kinds of files. In the implementation of this research, XDoclet for Hibernate is used to generate the mapping files for Hibernate *Configuration* to free the author from writing these files for all the data objects separately. It also frees the author from worrying that the deployment meta-data might be outdated. Please see Appendix F for implementation examples.

APPENDIX C. XML Parser

In the implementation of this research, XML is used for data exchange between the data management system and other application programs. To manipulate the XML file, we need an XML parser as shown in Figure 78. There are two widely used APIs for XML parsing and browsing: the Document Object Model (DOM) [137] and the Simple API for XML (SAX) [138].

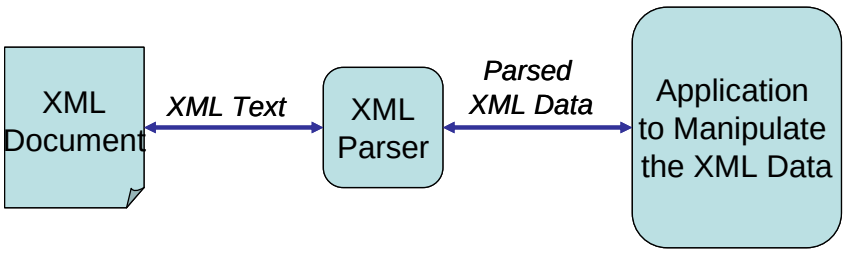

Figure 78. XML Parser

XML DOM is a W3C standard. XML DOM is platform and language independent. It defines a standard set of objects for XML and a standard way to access and manipulate XML documents. Basically, the XML DOM is a tree-based parser. It views an XML file as a tree structure of elements embedded within other elements. All these elements can be accessed through the DOM tree. The contents of these elements can be managed by the DOM.

SAX is a "de facto" standard. It was a Java-only API originally. The current version supports other programming languages such as C, C++, Perl, etc. SAX takes a very different approach from DOM. It is an event driven parser. It tells the application

what is in the XML document by notifying the application of a serialized stream of parsing events.

Each of these two parsing approaches has it own advantages and disadvantages. Compare to DOM, SAX has a storage size advantage. DOM keeps the whole XML document in memory, which is a heavy weight approach, whereas SAX reads an XML file and presents it as an event stream. This makes SAX good for processing large XML file. However, DOM supports instance and random access.

In the implementation, DOM is used for manipulating the XML file due to the fact that the file is not so large that it makes DOM an unappreciated approach. Xerces Java Parser developed by Apache [110], supports both DOM and SAX, and is used in the implementation.

APPENDIX D. MySQL

In the implementation of this research, MySQL is used as the relational database management system. MySQL is a popular, open source, easy to use SQL database management system. It is a product of the commercial company MySQL AB which was established in Sweden [139]. The software is written in C and C++, and can be used on many different platforms such as Windows, Sun Solaris, and Linux. It is client/server system that consists of a multi-threaded SQL server. The server supports different backbbones, several different client programs and libraries, administration tools, and APIs for C, C++, Java, Eiffel, Perl Tcl etc. The MySQL Server is an embedded multi-threaded library that can be linked to applications for a smaller, easier-to-manage product. However, it supports terabyte size databases. This is one of the main reasons why it is used in the implementation of this research. Java client programs that use JDBC connections are support by MySQL through the Connector/J interface which communicates directly with the MySQL server using the MySQL protocol. The security of MySQL is achieved through a privilege and password system that allows host based verification, and password traffic is encrypted when connecting to a server.

APPENDIX E. Implementation of the Object Oriented Data Model

In order to show how the classes and their relationships in the object-oriented data model are implemented, source codes of the "Variable" class, the "SystemComponent" class, and the "VC" class are given in this appendix. The implementation of other classes and the relationship among them are rather similar. These codes are written in Java. The object-relational mapping is embedded in the comments of these codes. The XDoclet package processes these comments and generates the Hibernate configuration files automatically. Based upon these configuration files, Hibernate stores the data objects into the MySQL relational database.

E.1 The "SystemComponent" Class

```
/*
 * Created on Jul 13, 2005 by Zhijie Lu (gte718q@mail.gatech.edu)
 */

package oad.pobj;
import java.util.*;
import oad.util.Helper;

/**
 * @author Zhijie
 *
 * @hibernate.class
 *            table="system_component"
 */
public class SystemComponent {

       private String id;
       private String name;
       private String description;

       //relationships
       //1. relationship with the "Variable" class, one-to-many
       private Set vcs;
       //2. relationship with the "GeometryConfiguration" class, many-to-one
       private GeometryConfiguration geometryConfiguration;
       //3. relationship with the "NonCompositeComponent" class, composition
       //private Set nonCompositeComponents = new HashSet();
       //4. relationship with the "CompositeComponent" class, composition
       private CompositeSystemComponent compositeSystemComponent;
```

```
/*
 * This constructor is protected
 */
protected SystemComponent(){
	vcs=new HashSet();
}

/**
 * @param id - set the id
 */
public void setId(String id){
	this.id=id;
}
/**
 * @return id - return the id
 *
 * @hibernate.id
 *            column="id"
 *            generator-class="uuid.hex"
 *            length="32"
 */
public String getId(){
	return this.id;
}

/**
 * @param name - set the name
 */
public void setName(String name){
	this.name=name;
}
/**
 * @return name - return the name
 * @hibernate.property
 *            column="name"
 */
public String getName(){
	return this.name;
}

/**
 * @param description - set the description
 */
public void setDescription(String description){
	this.description=description;
}
/**
 * @return description - return the description
 * @hibernate.property
 *            column="description"
 */
public String getDescription(){
	return this.description;
}

/**
 * Next three methods will do nothing for composite components
 * and will be overriden by subclass
 */
public void add(SystemComponent sc){
}
```

```
public Iterator getIterator(){
	return null;
}

public void remove(SystemComponent sc){
}

//relationships
//1. relationship with the "Variable" class, one-to-many
//2. relationship with the "GeometryConfiguration" class, many-to-one

/**
 * @return Returns the geometryConfiguration.
 * @hibernate.many-to-one
 *            cascade="all"
 *            column="geometry_configuration_id"
 */
public GeometryConfiguration getGeometryConfiguration() {
	return geometryConfiguration;
}
/**
 * @param geometryConfiguration The geometryConfiguration to set.
 */
public void setGeometryConfiguration(
            GeometryConfiguration geometryConfiguration) {
	this.geometryConfiguration = geometryConfiguration;
}

/**
 * @return Returns the compositeSystemComponent.
 * @hibernate.many-to-one
 *            cascade="all"
 *            column="composite_system_component_id"
 */
public CompositeSystemComponent getCompositeSystemComponent() {
	return compositeSystemComponent;
}
/**
 * @param compositeSystemComponent The compositeSystemComponent to set.
 */
public void setCompositeSystemComponent(
            CompositeSystemComponent compositeSystemComponent) {
	this.compositeSystemComponent = compositeSystemComponent;
}

/**
 * @return Returns the vcs.
 * @hibernate.set
 *            cascade="all"
 *            inverse=true
 *            lazy=true
 *            order-by="id asc"
 * @hibernate.collection-one-to-many
 *            class="oad.pobj.VC"
 * @hibernate.collection-key
 *            column="system_component_id"
 */
public Set getVCs() {
	return vcs;
}
/**
 * @param vas The vcs to set.
 */
```

```
	public void setVCs(Set vcs) {
		this.vcs = vcs;
	}
	/**
	 * add a variable to the variables Set
	 */
	public void addVC (VC vc){
		this.vcs.add(vc);
	}
	public Iterator getVCIterator(){
		return this.vcs.iterator();
	}
	public void print(){
		Helper.println("[SystemComponent]");
		Helper.println("name:"+this.name);
		Helper.println("desc:"+this.description);
		Helper.println("[Variable list]");

		TreeSet treeVCs=new TreeSet(vcs);
		Iterator it=treeVCs.iterator();

		while (it.hasNext()){
			VC vc=(VC)it.next();
			String out=vc.getVariable().getName()+"=";
			List list=vc.getVariable().getValues();
			Iterator it2=list.iterator();
			if(it2.hasNext()) out=out+it2.next();
			while (it2.hasNext()){
				out=out+", "+it2.next();
			}
			Helper.println(out);

			Helper.println("\tdesc:
					"+vc.getVariable().getDescription());
			Helper.println("\tunit: "+vc.getVariable().getUnit());
		}
	}
}
```

E.2 The "Variable" Class

```
/*
 * Created on Jul 13, 2005 by Zhijie Lu (gte718q@mail.gatech.edu)
 */

package oad.pobj;
import java.util.*;

/**
 * @author
 * @hibernate.class
 *            table="variable"
 */
public class Variable {

	private String id;
	private String name;
	private String description;
```

```
private List values;

private String valueType;
//private String defaultValue;
//private List defaultValues;
//private String value;//all value are double, need to wrapper to
                       handle the transform to other datatype
private String unit;
private String min; //lower limit
private String max; //upper limit
//private boolean controlVariable;
//private boolean analysisResult;
//private String obtainedFrom;

//relationships
//1. relationship with the "Requirement" class, many-to-one
private Set vrs;
//2. relationship with the "Component" class, many-to-one
private Set vcs;
//3. relationship with the "Analysis" class, many-to-one
private Set vas;

//"DisciplinaryAnalysis", "MissionAnalysis",and "optimization"
 //are subclasses of the "analysis" class,
//we don't need to construct with them seperately

public Variable(){
       vrs=new HashSet();
       vcs=new HashSet();
       vas=new HashSet();
       values=new ArrayList();
       //defaultValues= new ArrayList();
}

/**
 * @param id - set the id
 */
public void setId(String id){
       this.id=id;
}

/**
 * @return id - return the id
 * @hibernate.id
 *            column="id"
 *            generator-class="uuid.hex"
 *            length="32"
 */
public String getId(){
       return this.id;
}

/**
 * @param name - set the name;
 */
public void setName(String name){
       this.name=name;
}

/**
 * @return name - return the name
 * @hibernate.property
```

```
 *            column="name"
 *            sort="natural"
 */
public String getName(){
      return this.name;
}

/**
 * @param description - set the description
 */
public void setDescription(String description){
      this.description=description;
}

/**
 * @return description - return the descriptoin
 * @hibernate.property
 *            column="description"
 */
public String getDescription(){
      return this.description;
}

/**
 * @return Returns the vals.
 * @hibernate.list
 *            table="variable_values"
 *            cascade="save-update"
 * @hibernate.collection-element
 *            type="string"
 *            column="val"
 * @hibernate.collection-index
 *            column="idx"
 * @hibernate.collection-key
 *            column="variable_id"
 **/
public List getValues() {
      return values;
}

/**
 * @param values The values to set.
 */
public void setValues(List values) {
      this.values = values;
}

public void addValue(String value){
      this.values.add(value);
}

public Iterator getValuesIterator(){
      return this.values.iterator();
}

/**
 * @return Returns the valueType.
 * @hibernate.property
 *            column="value_type"
 */
public String getValueType() {
      return valueType;
}
```

```
/**
 * @param valueType The valueType to set.
 */
public void setValueType(String valueType) {
	this.valueType = valueType;
}

/**
 * @return Returns the vals.
 * @hibernate.list
 *            table="variable_values"
 *            cascade="save-update"
 * @hibernate.collection-element
 *            type="string"
 *            column="default_val"
 **/
/*
public List getDefaultValues() {
	return values;
}
*/

/**
 * @param values The values to set.
 */
/*
public void setDefaultValues(List values) {
	this.values = values;
}

public void addDefaultValue(String value){
	this.values.add(value);
}

public Iterator getDefaultValuesIterator(){
	return this.values.iterator();
}
*/

/**
 * @return Returns the defaultValue.
 * @hibernate.property
 *            column="defaut_value"
 */
/*
public String getDefaultValue() {
	return defaultValue;
}
*/

/**
 * @param defaultValue The defaultValue to set.
 */
/*
public void setDefaultValue(String defaultValue) {
	this.defaultValue = defaultValue;
}
*/

/**
 * @param unit - set the unit
 */
```

```
public void setUnit(String unit){
	this.unit=unit;
}

/**
 * @return unit - return the unit
 * @hibernate.property
 *            column="unit"
 */
public String getUnit(){
	return this.unit;
}

/**
 * @param upperlimit - set the upperlimit
 */
public void setMax(String max){
	this.max=max;
}

/**
 * @return upperlimit - return the upperlimit
 * @hibernate.property
 *            column="upper_limit"
 */
public String getMax(){
	return this.max;
}

/**
 * @param lowerlimit - set the lowerlimit
 */
public void setMin(String min){
	this.min=min;
}

/**
 * @return lowerlimit - return the lowerlimit
 * @hibernate.property
 *            column="lower_limit"
 */
public String getMin(){
	return this.min;
}


//relationships
//1. relationship with the "Requirement" class, many-to-one

/**
 * @return Returns the vrs.
 * @hibernate.set
 *            cascade="all"
 *            inverse="true"
 *            lazy="true"
 * @hibernate.collection-one-to-many
 *            class="oad.pobj.VR"
 * @hibernate.collection-key
 *            column="variable_id"
 */
public Set getVRs() {
	return vrs;
}
```

```
	/**
	 * @param vrs The vrs to set.
	 */
	public void setVRs(Set vrs) {
		this.vrs = vrs;
	}

	/**
	 * @return Returns the vas.
	 * @hibernate.set
	 *            cascade="all"
	 *            inverse="true"
	 *            lazy="true"
	 * @hibernate.collection-one-to-many
	 *            class="oad.pobj.VA"
	 * @hibernate.collection-key
	 *            column="variable_id"
	 */
	public Set getVAs() {
		return vas;
	}

	/**
	 * @param vas The vas to set.
	 */
	public void setVAs(Set vas) {
		this.vas = vas;
	}

	/**
	 * @return Returns the vcs.
	 * @hibernate.set
	 *            cascade="all"
	 *            inverse="true"
	 *            lazy="true"
	 * @hibernate.collection-one-to-many
	 *            class="oad.pobj.VC"
	 * @hibernate.collection-key
	 *            column="variable_id"
	 */
	public Set getVCs() {
		return vcs;
	}

	/**
	 * @param vcs The vcs to set.
	 */
	public void setVCs(Set vcs) {
		this.vcs = vcs;
	}
}
```

E.3 The “VC” Class

```
/*
 * Created on Jul 25, 2005 by Zhijie Lu (gte718q@mail.gatech.edu)
 */
```

```
package oad.pobj;


/**
 * @author Zhijie Lu
 * @hibernate.class
 *            table="vc"
 *
 */
public class VC implements Comparable{

	private String id;
	private SystemComponent component;
	private Variable variable;

	/**
	 *
	 */
	public VC() {
		//values=new ArrayList();
	}

	/**
	 * @return Returns the id.
	 * *@hibernate.id
	 *            column="id"
	 *            generator-class="uuid.hex"
	 *            length="32"
	 */
	public String getId() {
		return id;
	}

	/**
	 * @param id The id to set.
	 */
	public void setId(String id) {
		this.id = id;
	}

	/**
	 * @return Returns the component.
	 * @hibernate.many-to-one
	 *            cascade="none"
	 *            column="system_component_id"
	 */
	public SystemComponent getSystemComponent() {
		return component;
	}

	/**
	 * @param componet to set.
	 */
	public void setSystemComponent(SystemComponent sc) {
		this.component = sc;
	}

	/**
	 * @return Returns the variable.
	 * @hibernate.many-to-one
	 *            cascade="save-update"
	 *            column="variable_id"
	 */
```

```
	public Variable getVariable() {
		return variable;
	}

	/**
	 * @param variable The variable to set.
	 */
	public void setVariable(Variable variable) {
		this.variable = variable;
	}
	public int compareTo(Object obj){
		VC vc=(VC)obj;
		return this.variable.getName().compareTo(vc.variable.getName());
	}
}
```

APPENDIX F. The Interface of the Data Management System

A command line interface (CLI) is used for the prototype data management system. A CLI is a straightforward command interface. Through the interface, lines of command text can be entered. Then after pressing the "Enter" key the commands are executed. The output is also received as text. It is not totally replaced by a GUI because compared to a GUI, a CLI is sometimes easier, more direct and efficient.

A reusable open source Java toolkit, the CLI component of Apache Jakarta Commons [109] is used in the implementation. It provides a simple and easy to use API for working with the command line arguments and options. Figure 79 is a screenshot of the interface of the data management system.

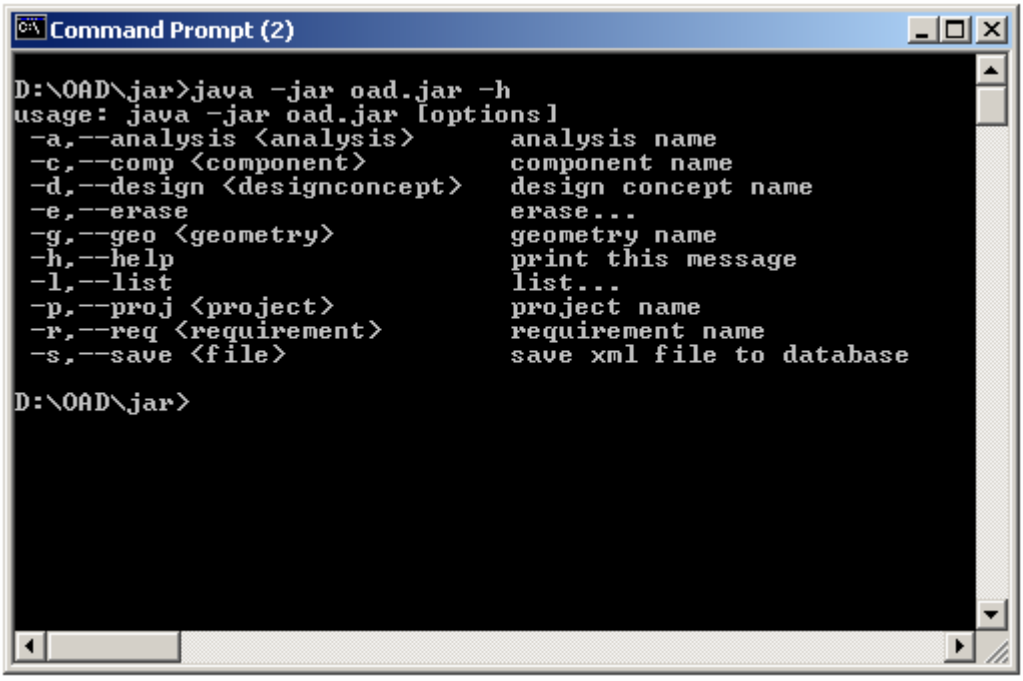

Figure 79. The Interface of the Data Management System

APPENDIX G. The XML DTD File

The complete DTD for data exchange XML files is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by Zhijie
Lu (Georgia Institute of Technology) -->
<!ELEMENT project (requirement | designconcept | variable)*>
<!ELEMENT requirement (vr)*>
<!ELEMENT designconcept (geoconfig | optimization | discipanalysis |
missionanalysis)*>
<!ELEMENT geoconfig (comsyscomp | leafsyscomp)*>
<!ELEMENT comsyscomp (comsyscomp | leafsyscomp)*>
<!ELEMENT leafsyscomp (vc)*>
<!ELEMENT optimization (analysis)>
<!ELEMENT discipanalysis (analysis)>
<!ELEMENT missionanalysis (analysis)>
<!ELEMENT analysis (extdatasrc | va)*>
<!ELEMENT variable EMPTY>
<!ELEMENT va EMPTY>
<!ELEMENT vc EMPTY>
<!ELEMENT vr EMPTY>
<!ELEMENT extdatasrc (#PCDATA)>
<!ATTLIST project
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST requirement
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST designconcept
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST geoconfig
ID #REQUIRED
CDATA #IMPLIED
>
<!ATTLIST comsyscomp
ID #REQUIRED
CDATA #IMPLIED
>
<!ATTLIST leafsyscomp
```

```
ID #REQUIRED
CDATA #IMPLIED
>
<!ATTLIST analysis
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST va
IDREF #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST vr
IDREF #REQUIRED
>
<!ATTLIST vc
IDREF #REQUIRED
>
<!ATTLIST variable
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
<!ATTLIST extdatasrc
ID #REQUIRED
CDATA #IMPLIED
CDATA #IMPLIED
CDATA #IMPLIED
>
```

APPENDIX H. Integrated Design Environments in *ModelCenter*

Screen shots of the integrated design environments with or without the support of the data management system in *ModelCenter* for both case studies are provided in this appendix. With the support of a data management system, the design and analysis process is simplified.

H.1 Case Study I: A Notional Conventional Aircraft

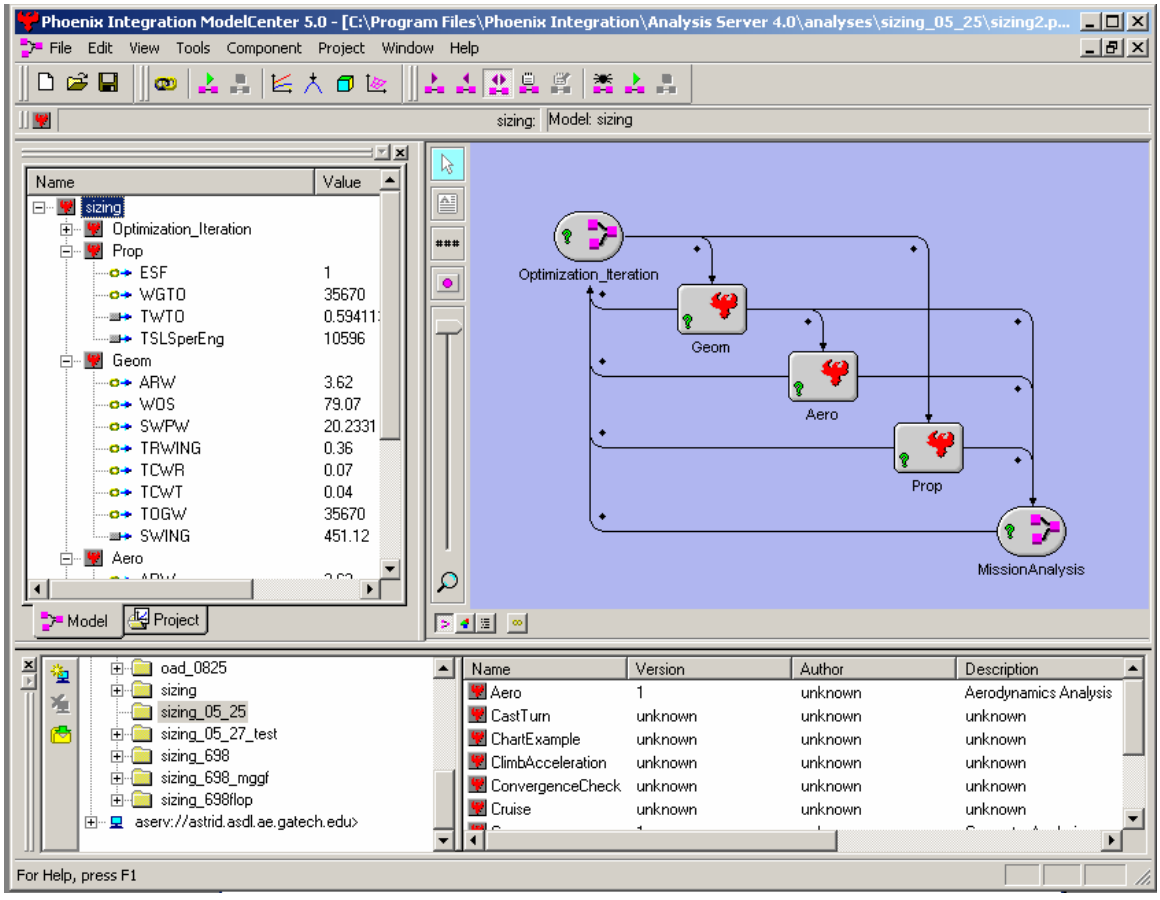

Figure 80. Case Study I: Integrated Design Environment in *ModelCenter* without the Support of the Data Management System

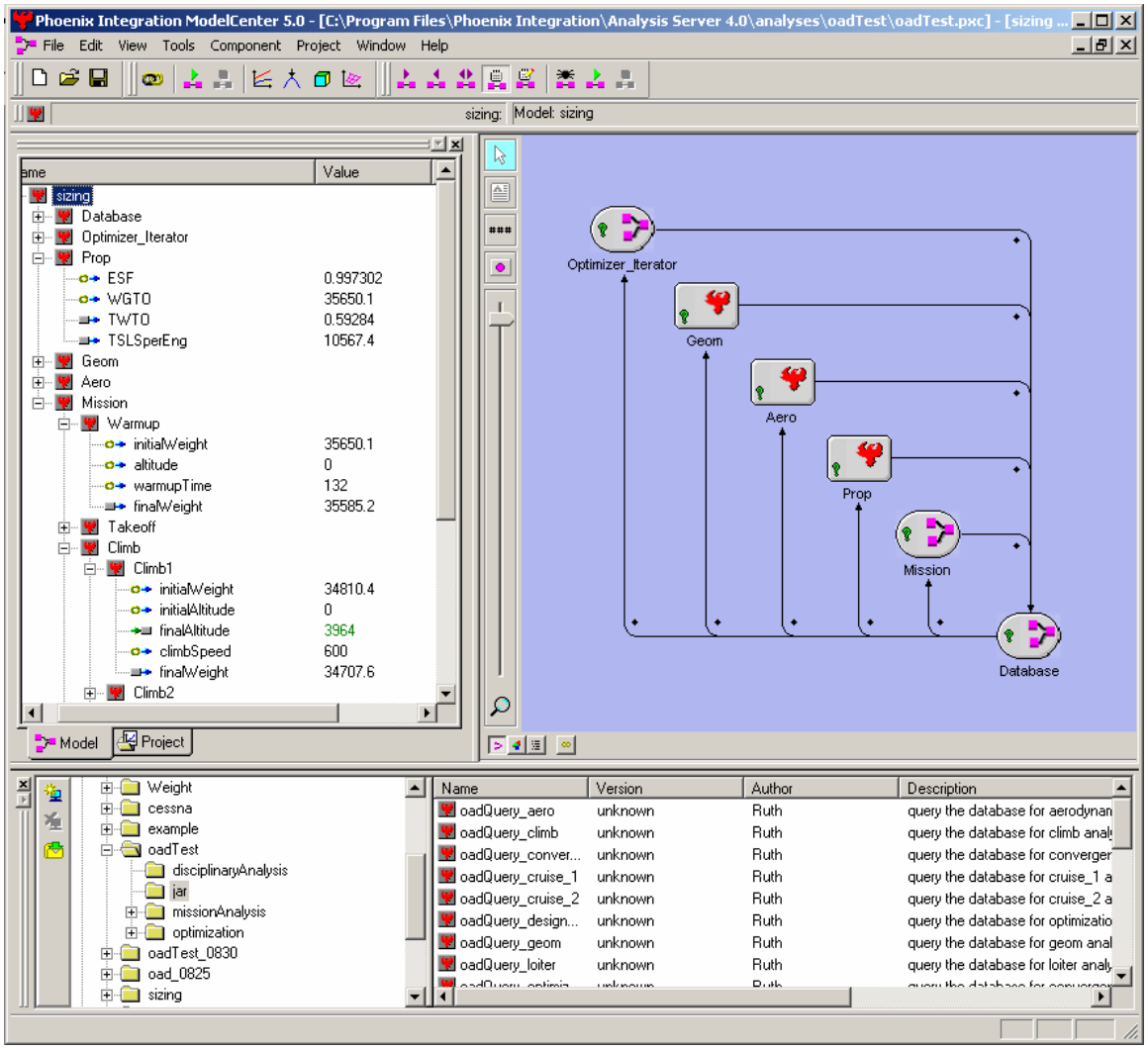

Figure 81. Case Study I: Integrated Design Environment in *ModelCenter* with the Support of the Data Management System

H.2 Case Study II: An Unconventional Aircraft

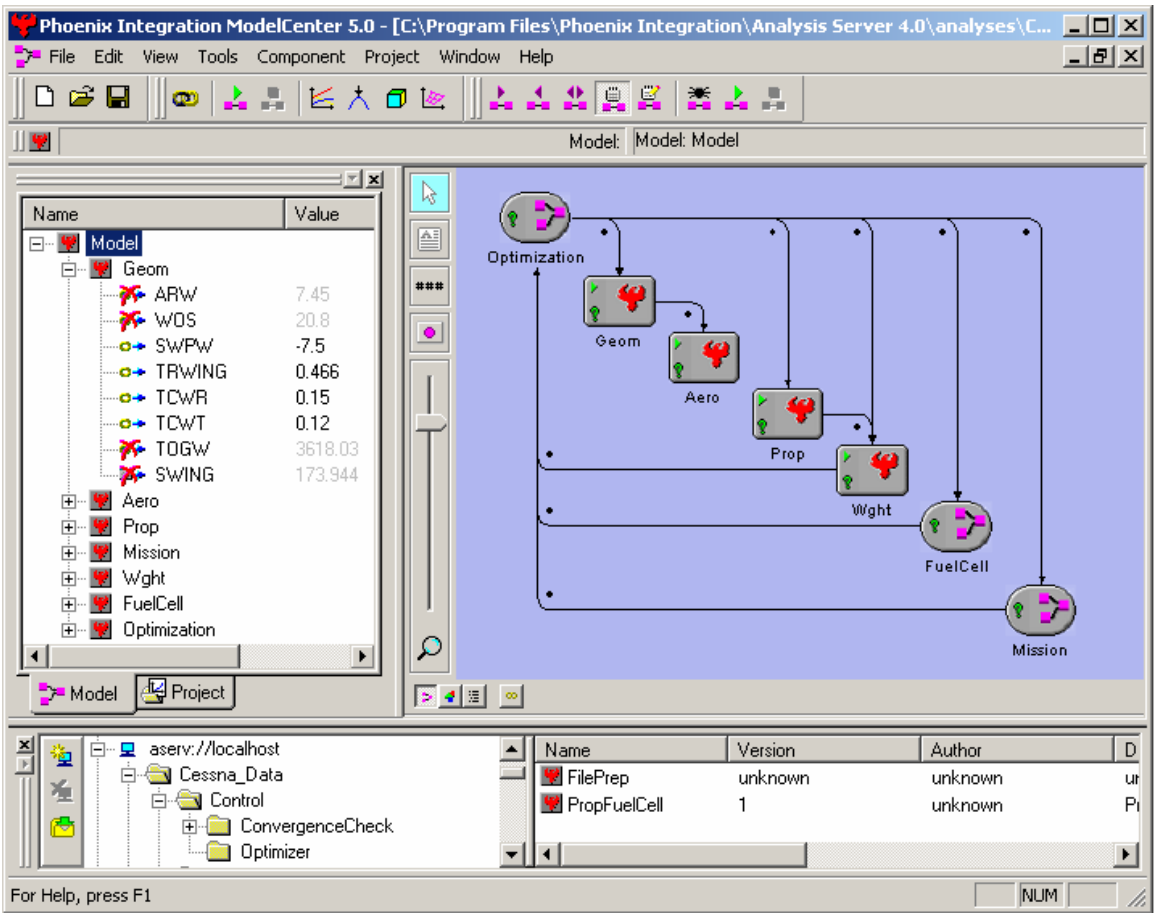

Figure 82. Case Study II: Integrated Design Environment in *ModelCenter* without the Support of the Data Management System

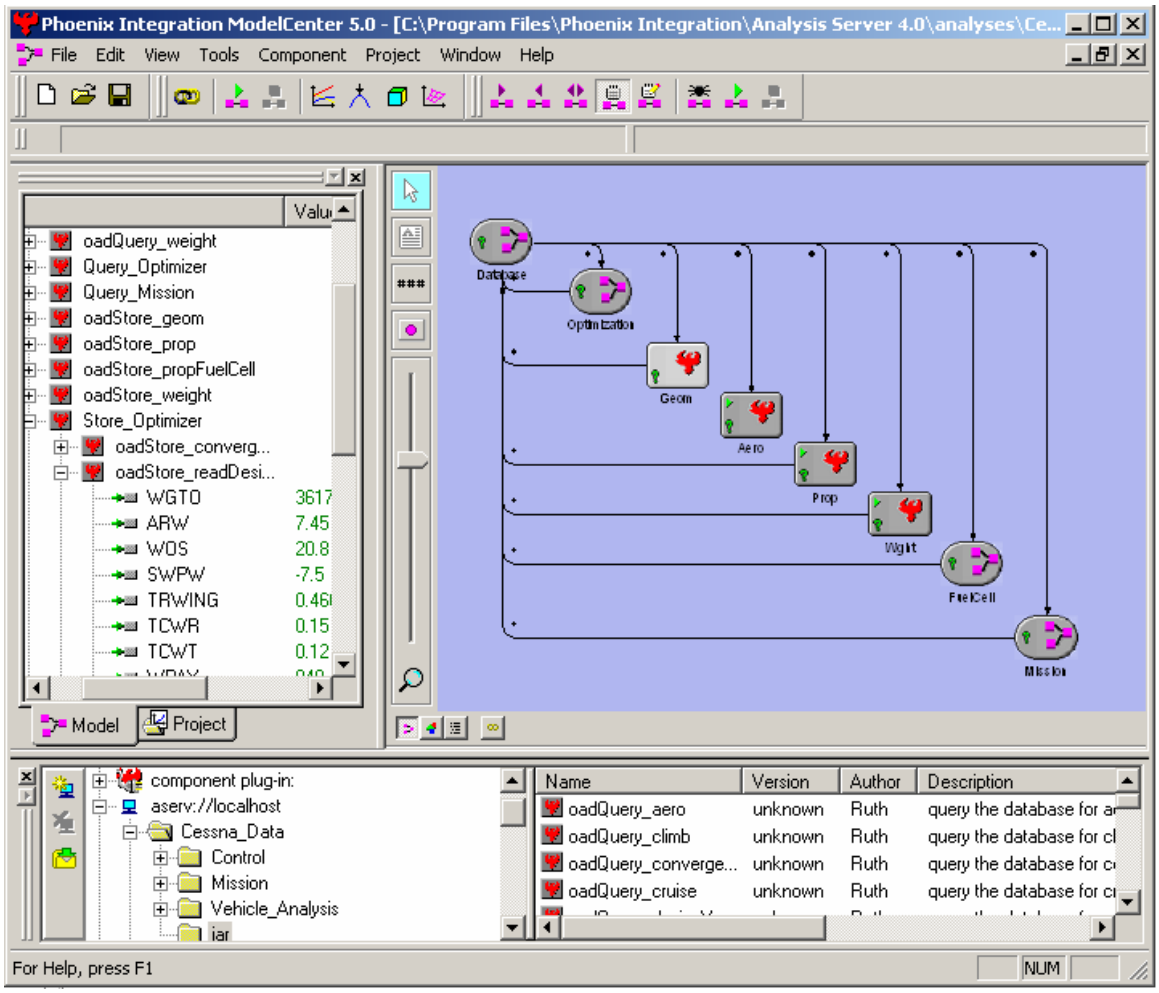

Figure 83. Case Study II: Integrated Design Environment in *ModelCenter* with the Support of the Data Management System

APPENDIX I. Design Processes

The Visual Basic scripts called Scheduler in *ModelCenter* are used to automate the design process.

I.1 Case Study I: A Notional Conventional Aircraft

```
set readDesignVariables=app.getComponent(("sizing.Database.
oadStore_readDesignVariables")

set designVariables_query=app.getComponent("sizing.Database.
oadQuery_designVariables")

set designVariables=app.getComponent("sizing.Optimizer_
Iterator.DesignVariables")

set designVariables_store=app.getComponent("sizing.Database.
oadStore_designVariables")

set convergence_query=app.getComponent("sizing.Database.
oadQuery_convergenceCheck")

set convergence=app.getComponent("sizing.Optimizer_Iterator.
ConvergenceCheck")

set convergence_store=app.getComponent("sizing.Database.
oadStore_convergenceCheck")

set geom_query=app.getComponent("sizing.Database.oadQuery_
geom")

set geom=app.getComponent("sizing.Geom")

set geom_store=app.getComponent("sizing.Database.
oadStore_geom")

set aero_query=app.getComponent("sizing.Database.
oadQuery_aero")

set aero=app.getComponent("sizing.Aero")

set aero_store=app.getComponent("sizing.Database.
oadStore_aero")

set prop_query=app.getComponent("sizing.Database.
oadQuery_prop")

set prop=app.getComponent("sizing.Prop")

set prop_store=app.getComponent("sizing.Database.
```

```
oadStore_prop")

set warmup_query=app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_warmup")

set warmup=app.getComponent("sizing.Mission.Warmup")

set warmup_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_warmup")

set takeoff_query=app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_takeoff")

set takeoff=app.getComponent("sizing.Mission.Takeoff")

set takeoff_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_takeoff")

set climb_query=app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_climb")

set climb1=app.getComponent("sizing.Mission.Climb.Climb1")

set climb2=app.getComponent("sizing.Mission.Climb.Climb2")

set climb3=app.getComponent("sizing.Mission.Climb.Climb3")

set climb4=app.getComponent("sizing.Mission.Climb.Climb4")

set climb5=app.getComponent("sizing.Mission.Climb.Climb5")

set climb6=app.getComponent("sizing.Mission.Climb.Climb6")

set climb7=app.getComponent("sizing.Mission.Climb.Climb7")

set climb8=app.getComponent("sizing.Mission.Climb.Climb8")

set climb9=app.getComponent("sizing.Mission.Climb.Climb9")

set climb10=app.getComponent("sizing.Mission.Climb.Climb10")

set climb_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_climb")

set cruise1_query =app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_cruise_1")

set cruise1_c1=app.getComponent("sizing.Mission.Cruise_1.Cruise1")

set cruise1_c2=app.getComponent("sizing.Mission.Cruise_1.Cruise2")

set cruise1_c3=app.getComponent("sizing.Mission.Cruise_1.Cruise3")

set cruise1_c4=app.getComponent("sizing.Mission.Cruise_1.Cruise4")

set cruise1_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_cruise_1")

set turn_query = app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_turn")

set turn=app.getComponent("sizing.Mission.CastTurn")
```

```
set turn_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_turn")

set cruise2_query = app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_cruise_2")

set cruise2_c1=app.getComponent("sizing.Mission.Cruise_2.Cruise1")

set cruise2_c2=app.getComponent("sizing.Mission.Cruise_2.Cruise2")

set cruise2_c3=app.getComponent("sizing.Mission.Cruise_2.Cruise3")

set cruise2_c4=app.getComponent("sizing.Mission.Cruise_2.Cruise4")

set cruise2_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_cruise_2")

set loiter_query = app.getComponent("sizing.Database.
oadQueryStore_mission.oadQuery_loiter")

set loiter1=app.getComponent("sizing.Mission.Loiter.Loiter1")

set loiter2=app.getComponent("sizing.Mission.Loiter.Loiter2")

set loiter3=app.getComponent("sizing.Mission.Loiter.Loiter3")

set loiter4=app.getComponent("sizing.Mission.Loiter.Loiter4")

set loiter_store=app.getComponent("sizing.Database.
oadQueryStore_mission.oadStore_loiter")

sub run
   readDesignVariables.run
   designVariables.run
   designVariables_store.run

   do
          geom_query.run
          geom.run
          geom_store.run

          aero_query.run
          aero.run
          aero_store.run

          prop_query.run
          prop.run
          prop_store.run

          warmup_query.run
          warmup.run
          warmup_store.run

          takeoff_query.run
          takeoff.run
          takeoff_store.run

          climb_query.run
          climb1.run
          climb2.run
          climb3.run
          climb4.run
          climb5.run
```

```
            climb6.run
            climb7.run
            climb8.run
            climb9.run
            climb10.run
            climb_store.run

            cruise1_query.run
            cruise1_c1.run
            cruise1_c2.run
            cruise1_c3.run
            cruise1_c4.run
            cruise1_store.run

            turn_query.run
            turn.run
            turn_store.run

            cruise2_query.run
            cruise2_c1.run
            cruise2_c2.run
            cruise2_c3.run
            cruise2_c4.run
            cruise2_store.run

            loiter_query.run
            loiter1.run
            loiter2.run
            loiter3.run
            loiter4.run
            loiter_store.run

            convergence_query.run
            convergence.run
            convergence_store.run
    loop while

app.getValue("sizing.Optimizer_Iterator.ConvergenceCheck.
WfuelDiffAbsolute") >= 1.00

end sub
```

I.2 Case Study II: An Unconventional Aircraft

```
set readDesignVariables = app.getComponent("Model.Database.Store_optimizer.
oadStore_readDesignVariables")

set designVariables_query = app.getComponent("Model.Database.Query_optimizer.
oadQuery_designVariables")

set designVariables = app.getComponent("Model.Optimization.Optimizer")

set designVariables_store = app.getComponent("Model.Database.Store_optimizer.
oadStore_designVariables")
```

```
set convergence_query = app.getComponent("Model.Database.Query_optimizer.
oadQuery_convergenceCheck")

set convergence = app.getComponent("Model.Optimization.ConvergenceCheck")

set convergence_store = app.getComponent("Model.Database.Store_optimizer.
oadStore_convergenceCheck")

set geom_query = app.getComponent("Model.Database.oadQuery_geom")

set geom = app.getComponent("Model.Geom")

set geom_store = app.getComponent("Model.Database.oadStore_geom")

set aero_query = app.getComponent("Model.Database.oadQuery_aero")

set aero = app.getComponent("Model.Aero")

set prop_query = app.getComponent("Model.Database.oadQuery_prop")

set prop = app.getComponent("Model.Prop")

set prop_store = app.getComponent("Model.Database.oadStore_prop")

set fuelCell_query = app.getComponent("Model.Database.oadQuery_propFuelCell")

set fuelCell = app.getComponent("Model.FuelCell.PropFuelCell")

set fuelCellPrep = app.getComponent("Model.FuelCell.FilePrep")

set fuelCell_store = app.getComponent("Model.Database.oadStore_propFuelCell")

set weight_query = app.getComponent("Model.Database.oadQuery_weight")

set weight = app.getComponent("Model.Wght")

set weight_store = app.getComponent("Model.Database.oadStore_weight")

set warmup_query = app.getComponent("Model.Database.Query_mission.
oadQuery_warmup")

set warmup = app.getComponent("Model.Mission.Warmup")

set warmup_store = app.getComponent("Model.Database.Store_mission.
oadStore_warmup")

set takeoff_query = app.getComponent("Model.Database.Query_mission.
oadQuery_takeoff")
set takeoff = app.getComponent("Model.Mission.Takeoff")

set takeoff_store = app.getComponent("Model.Database.Store_mission.
oadStore_takeoff")

set climb_query = app.getComponent("Model.Database.Query_mission.
oadQuery_climb")

set climb = app.getComponent("Model.Mission.ClimbAcceleration")

set climb_store = app.getComponent("Model.Database.Store_mission.
oadStore_climb")

set cruise_query = app.getComponent("Model.Database.Query_mission.
oadQuery_cruise")
```

```
set cruise = app.getComponent("Model.Mission.Cruise")

set cruise_store = app.getComponent("Model.Database.Store_mission.
oadStore_cruise")

set loiter_query = app.getComponent("Model.Database.Query_mission.
oadQuery_loiter")

set loiter = app.getComponent("Model.Mission.Loiter")

set loiter_store = app.getComponent("Model.Database.Store_mission.
oadStore_loiter")

DFP = app.getValue("Model.Optimization.Optimizer.dissiptiveFuelPercent")

sub run

   readDesignVariables.run
   designVariables.run
   designVariables_store.run

   do
          geom_query.run
          geom.run
          geom_store.run

          aero_query.run
          aero.run

          if (DFP > 0) then
                 prop_query.run
                 prop.run
                 prop_store.run
          end if

          if (DFP < 1) then
                 fuelCell_query.run
                 fuelCell.run
                 fuelCellPrep.run
                 fuelCell_store.run
          end if

          weight_query.run
          weight.run
          weight_store.run

          warmup_query.run
          warmup.run
          warmup_store.run

          takeoff_query.run
          takeoff.run
          takeoff_store.run

          climb_query.run
          climb.run
          climb_store.run

          cruise_query.run
          cruise.run
          cruise_store.run
```

```
            loiter_query.run
            loiter.run
            loiter_store.run

            convergence_query.run
            convergence.run
            convergence_store.run

            WfuelDiffAbsolute = abs(app.getValue("Model.
            ConvergenceCheck.WfuelAvail")-app.getValue("Model.
            ConvergenceCheck.WfuelRequired"))

    loop while WfuelDiffAbsolute >= 1.00
```

end sub

APPENDIX J. Wrappers

ModelCenter wrappers used to integrate the database with other analysis programs in *ModelCenter*.

J.1 Case Study I: A Notional Conventional Aircraft

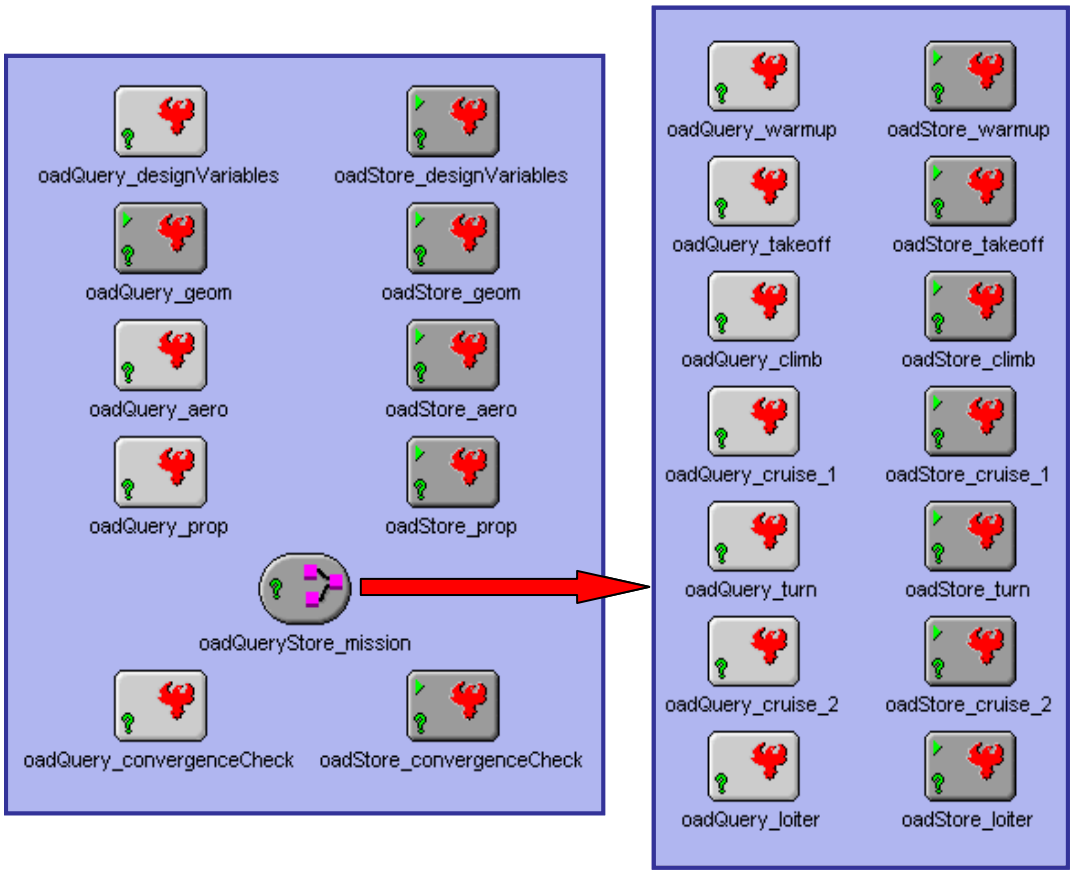

Figure 84. Case Study I: *ModelCenter* Wrappers for the Integration of the Data Management System

J.2 Case Study II: An Unconventional Aircraft

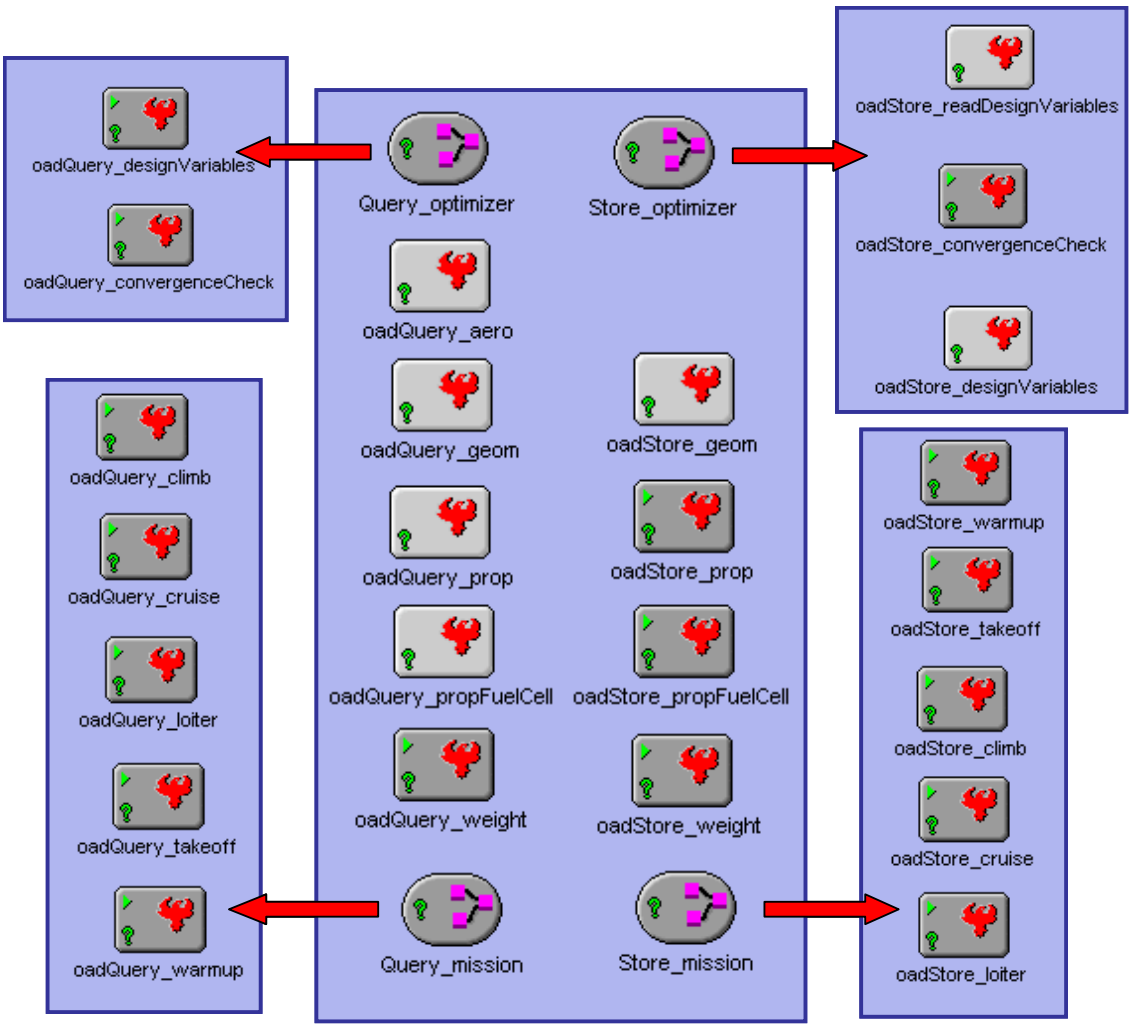

Figure 85. Case Study II: *ModelCenter* Wrappers for the Integration of the Data Management System

REFERENCES

1. *Research Opportunities in Engineering Design: NSF Strategic Planning Workshop Final Report*. 1996, National Science Foundation.
2. *IEEE Standard Glossary of Software Engineering Terminology*. 1990, IEEE Standard 610.12-1990, ISBN 1-55937-067-X.
3. Ballard, R., *Toward Knowledge Based Computing*, in *DestinationKM.com, Steve Barth*. 2000, http://www.destinationkm.com/articles/default.asp?ArticleID=354. Aug. 2005
4. Turing, A.M., *Computing Machinery and Intelligence.* MIND, 1950. **LIX**(236): p. 433-460.
5. Russell, S.J. and P. Morvig, *Artificial Intelligence: A Modern Approach*. 2nd ed. 2002: Prentice Hall. 1132.
6. Sobieszezanski-Sobieski, J. and R.T. Haftka. *Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments*. in *The 34th Aerospace Sciences Meeting and Exhibit*. 1996. Reno, NV.
7. Sobieszczanski-Sobieski, J., *Multidisciplinary Design Optimization: an Emerging, New Engineering Discipline*, in *Advances in Structural Optimization*, J. Herskovits, Editor. 1995, Kluwer Academic. p. 483-496.
8. Sobieszczanski-Sobieski, J., *Sensitivity Analysis and Multidisciplinary Optimization for Aircraft Design: Recent Advances and Results.* Journal of Aircraft, 1990. **27**(12): p. 993-1001.
9. Booch, G., *Object-Oriented Analysis and Design with Applications*. 2nd ed. 1994: Addison-Wesley Publishing Company.
10. *http://www.m-w.com/dictionary/information+science, Merriam-Webster Online Dictionary*. Aug. 2005.
11. Saracevic, T., *Information Science.* Journal of the American Society for Information Science, 1999. **50**(12): p. 1051-1063.
12. Hammarberg, R., *The cooked and the raw.* Journal of Information Science, 1981. **3**(6): p. 261-267.
13. Bates, M.J., *Information and knowledge: an evolutionary framework for information science.* Information Research, 2005. **10**(4): p. 239.
14. *Advanced Engineering Environments (Phase 1): Achieving the Vision*. 2000, National Academy of Engineering and National Research Council.
15. *Advanced Engineering Environments (Phase 2): Design in the New Millennium*. 2000, National Academy of Engineering and National Research Council.
16. McCullers, L.A., *FLight Optimization System, FLOPS, User's Guide, Release 5.94*. Dec 1998, NYMA, Inc.: Hampton, VA.
17. Wrenn, G.A., *An Indirect Method for Numerical Optimization Using the Kreisselmeier-Steinhase Function*. Mar 1989, NASA CR 4220.

18. Fiacco, A. and G.P. McCormick, *The Sequential Unconstrained Minimization Technique for Nonlinear Programming, A Primal - Dual Method.* Management Science, Jan 1964. **10**(2): p. 367-385.

19. GelHausen, P.A., M.D. Moore, and J.R. Gloudemans. *Overview of ACSYNT for Light Aircraft Design*. in *SAE World Aviation Conference*. 1995. Los Angeles, CA.

20. Vanderplaats, G.N., *CONMIN - A Fortran Program for Constrained Function Minimization - User's Manual*. Aug. 1973, NASA Ames Research Center and U.S. Army Air Mobility R&D Laboratory.

21. Lu, Z., et al. *Formulation and Test of an Object-Oriented Approach to Aircraft Sizing*. in *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Aug. 2004. Albany, New York.

22. Stephens, E.R., *LEGEND: Laboratory Environment for the Generation, Evaluation, and Navigation for Design*, Doctoral Dissertation, 1993, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA.

23. Hale, M.A., D.N. Mavris, and D.L. Carter. *The Implementation of a Conceptual Aerospace Systems Design and Analysis Toolkit*. in *The 4th World Aviation Congress and Exposition*. Oct 1999. San Franciso, CA.

24. Hale, M.A., J.I. Craig, and F. Mistree, *DREAMS and IMAGE: A Model and Computer Implementation for Concurrent, Life-Cycle Design of Complex Systems.* Concurrent Engineering: Research and Applications, June 1996. **4**(2): p. 171-186.

25. Hale, M., *An Open Computing Infrastructure that Facilitates Integrated Product and Process Development from a Decision-Based Perspective*, Doctoral Dissertation, July 1996, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA.

26. Hale, M. and D. Mavris. *Enabling Advanced Design Methods in an Internet-Capable Framework*. in *The World Aviation Congress and Exposition*. 1999: SAE Paper number 1999-01-5578.

27. Kolb, M., *An Investigation of Constraint-Based Component-Modeling for Knowledge Representation in Computer-Aided Conceptual Design*, Doctoral Dissertation, Jan. 1990, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

28. Holle, J.V., *The New SBA - Revisited*. 2004, Modeling and Simulation Information Analysis Center.

29. Hurwitz, M., *Information Integration via Navy LEAPS*. in *Enabling the 21st Century Acquisition Enterprise, 3rd Simulation Based Acquisition Conference*. 2001.

30. Binder, M., *Numerical Propulsion System Simulation Introduction. Multimedia CD*. 2001: Washington, DC.

31. Evans, A.L. and et al. *Numerical Propulsion System Simulation's National Cycle Program*. 1998.

32. Deutsch, M.-J. and J.S. Nichols, *Advanced Approach to Concept and Design Studies for Space Mission.* Astrophysics and Space Science, Sep. 2000. **273**: p. 201-206.

33. http://www.aero.org/publications/crosslink/winter2001/01.html, *the website of The Aerospace Corporation*. Jan. 2007.

34. Rumble, J.R. and V.E. Hampel, *Database Management in Science and Technology: A CODATA Sourcebook on the Use of Computers in Data Activities*. 1984: Elsevier Science Pub. Co.

35. Katz, R.H., *Information Management in Engineering Design*. Surveys in Computer Science. 1985, Heidelberg: Springer-Verlag.

36. Morris, K.C., et al., *Database Management Systems in Engineering*. 1992, National Institute of Standards and Technology: Gaithersburg, Maryland.

37. Karinthi, R., et al., *Promoting Concurrent Engineering through Information Sharing*. 1992, Concurrent Engineering Research Center: Morgantown, WV.

38. Peak, R.S. and R.E. Fulton. *Automating Routine Analysis in Electronic Packaging Using Product Model-Based Analytical Models (PBAMS), Part I: PBAM Overview*. in *ASME Winter Annual Meeting*. 1993. New Orleans.

39. Peak, R.S., et al., *Integrating Engineering Design and Analysis Using a Multi-Representation Approach.* Engineering with Computers, 1998. **14**(2): p. 93-114.

40. Hall, N.S. and R.E. Fulton. *Impact of Data Modeling and Database Implementation Methods on the optimization of Conceptual Aircraft Design*. in *ASME Engineering Information Management Symposium*. 1997. Sacramento, CA.

41. *Federal Information Processing Standards Publication 184: "Announcing the Standard for Integration Definition for Information Modeling (IDEF1X)"*. 1993.

42. *ISO 10303, Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 11: Description Methods: The EXPRESS Language Reference Manual, ISO TC 184/SC94*. 1994.

43. *http://www.steptools.com, the Homepage of STEP Tools, Inc.* Aug. 2005.

44. Loffredo, D., *Fundamentals of STEP Implementation*, STEP Tools, Inc.

45. Lin, R. and A.A. Afjeh, *An XML-based Integrated Database Model for Improving Multidisciplinary Aircraft Design.* Journal of Aerospace Computing, Information, and Communication, 2004. **1**(3): p. 154-172.

46. Kam, J.V. and P. Gage. *The Launch Vehicle Language (LVL) Data Model for Evaluating Reusable Launch Vehicle Concepts*. in *The 41st Aerospace Sciences Meeting & Exhibit*. Jan 2003. Reno, NV.

47. Li, Y., W. Shen, and H. Ghenniwa, *Collaborative and Proactive Data Agent for Distributed Design Environments.* Journal of Integrated Design and Process Science, 2003. **7**(2): p. 71-78.

48. Chandrasekaran, B., J.R. Josephson, and V.R. Benjamins, *What are Ontologies, and Why do We Need Them?* IEEE Intelligent Systems, 1999. **14**(1): p. 20-16.

49. Noy, N.F. and C.D. Hafner, *The State of the Art in Ontology Design: A Survey and Comparative Review.* AI Magazine, 1997. **18**(3): p. 53-74.

50. Daconta, M.C., L.J. Obrst, and K.T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. 2003: John Viley & Sons.

51. Gennari, J., et al., *The Evolution of Protégé: An Environment for Knowledge-Based Systems Development*. 2002.

52. *http://www.technosoft.com*, *the website of TechnoSoft, Inc.* June 2005

53. *http://phoenix-int.com*, *the website of Phoenix Integration, Inc*. June 2005.

54. *http://www.engineous.com*, *the website of Engineous Software, Inc.* June 2005.

55. Anderson, J.D., *Aircraft Performance and Design*. 1999: The McGraw-Hill Companies Inc.

56. http://www.incose.org, *the website of International Council on Systems Engineering*. Dec. 2006.

57. Torenbeek, E., *Synthesis of Subsonic Airplane Design: an Introduction to the Preliminary Design of Subsonic General Aviation and Transport Aircraft, with Emphasis on Layout, Aerodynamic design, Propulsion and Performance*. 1976, Netherlands: Delft University Press.

58. DeLaurentis, D.A., *A Probabilistic Approach to Aircraft Design Emphasizing Stability and Control Uncertainties*. Doctoral Dissertation, Dec. 1998, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA

59. Schrage, D.P. *Technology for Rotorcraft Affordability through Integrated Product/Process Development (IPPD)*. in *The American Helicopter Society 55th Annual Forum*. May 1999. Montreal, Canada.

60. Pugliese, J.J. *Design Synthesis*. in *The Aircraft Design Process Seminar*. 1971: Society of Aeronautical Weight Engineers/University of Texas - Arlington.

61. Mattingly, J.D., *Aircraft Engine Design*. AIAA Education Series, ed. J.S. Przemieniecki. 1987, New York, NY: American Institute of Aeronautics and Astronautics, Inc.

62. Raymer, D.P., *Aircraft Design: A Conceptual Approach*. 3rd ed. AIAA Education Series, ed. J.S. Przemienieck. 1999, Reston, VA: American Institute of Aeronautics and Astronautics, Inc.

63. Mavris, D.N., et al. *A Stochastic Approach to Multi-disciplinary Aircraft Analysis and Design*. in *The 36th Aerospace Sciences Meeting & Exhibit*. 1998. Reno, NV.

64. *http://www.jsf.mil, the F-35 Joint Strike Fighter Program*. Aug. 2005.

65. Mavris, D.N. and D. DeLaurentis. *An Integrated Approach to Military Aircraft Selection and Concept Evaluation*. in *The 1st AIAA Aircraft Engineering, Technology, and Operations Congress*. Sep. 1995. Los Angeles, CA: AIAA.

66. Hunt, V.D., *Enterprise Integration Sourcebook: the Integration of CALS, CE, TQM, PDES, RAMP, and CIM*. July 1991: Academic Press.

67. *DoD Integrated Product and Process Development Handbook*. 1998, Office of the Under Secretary of Defense, Washington, DC.

68. Taguchi, G., *Introduction to Quality Engineering: Designing Quality into Products and Processes*. 1986, American Supplier Institute.

69. Mavris, D.N., O. Bandte, and D.A. DeLaurentis, *Robust Design Simulation: A Probabilistic Approach to Multidisciplinary Design.* Journal of Aircraft, 1999. **36**(1): p. 298-307.

70. *www.foldoc.org, Online Dictionary of Computing*. 2004.

71. *FNC Resolution: Definition of "Internet", http://www.itrd.gov/fnc/Internet_res.html*. October 1995, the Federal Networking Council. Aug. 2005

72. Berners-Lee, T., *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. 1st ed. 1999: San Francisco: HarperSanFrancisco.

73. Stefik, M. and D. Bobrow, *Object-Oriented programming: Themes and Variations.* AI Magazine, 1986. **6**(4): p. 41.

74. Dahl, O.-H. and K. Nygaard, *SIMULA: an ALGOL-Based Simulation Language.* Communications of the ACM, 1966. **9**(9): p. 671-678.

75. Hunt, J., *Smalltalk and Object Orientation: An Introduction*. 1997: Springer-Verlag Telos.

76. Meyer, B., *Eiffel: Programming for Reusability and Extendibility.* ACM SIGPLAN Notices, 1987. **22**(2): p. 85-94.

77. Stroustrup, B., *The C++ Programming Language*. 3rd ed. 2000: Addison-Wesley Professional.

78. *http://java.sun.com, the website of the Java Programming Language*. June 2005.

79. Liberty, J., *Programming C#.* 3rd ed. 2003: O'Reilly Media, Inc.

80. Horstmann, C.S. and G. Cornell, *Core Java 2: Volume I - Fundamentals*. 4th ed. The Sun Microsoftsystems Press Java Series. 1999, Upper Saddle River, NJ: Prentice Hall PTR.

81. Horstmann, C.S. and G. Cornell, *Core Java 2: Volume II - Advanced Features*. 4th ed. The Sun Microsystems Press Java Series. 1999, Upper Saddle River, NJ: Prentice Hall PTR.

82. Fowler, M. and K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. 2003: Addison-Wesley Publishing Company.

83. Mats, L., *UML 2.0 & Model-Driven Development.* Dr. Dobb's Journal: Software Tools for the Professional Programmer, Aug 2003. **28**(8): p. 50, 3p.

84. http://www.omgsysml.org, *the website of OMG (2006) OMG Systems Modeling Language (OMG SysML)*. Dec. 2006.

85. OMG, *Final Adopted OMG SysML Specification (ptc/06-05-04)*. May 2006.

86. Friedenthal, S., A. Moore, and F. Steiner. *OMG Systems Modeling Language (OMG SysML) Turtorial*. in *INCOSE Intl. Symp.* 2006. Orlanda, FL.

87. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional Computing Series. 1995, Reading, MA: Addison-Wesley Professional. 395.

88. Marston, T., *The Model-View-Controller (MVC) Design Pattern for PHP*. 2005.

89. Connolly, T.M. and C.E. Begg, *Database Solutions: A Step-by-Step Approach to Building Database*. Vol. 2nd. 2000, Reading, Mass.: Addison-Wesley, Inc.

90. http://www.m-w.com. *Merriam-Webster Online Dictionary*. July 2004,

91. Elmasri, R. and S.B. Navathe, *Fundamentals of Database Systems*. 3rd edition ed. 2000, Reading, Mass.: Addison Wesley Longman, Inc.

92. Date, C.J., *The Database Relational Model: a Retrospective Review and Analysis: a Historical Account and Assessment of E.F. Codd's Contribution to the field of Database Technology*. 2001, Reading, MA: Addison-Wesley Publishing Company.

93. Date, C.J., *An Introduction to Database System*. 1995: Addison-Wesley Publishing Company.

94. Bertino, E. and L. Martino, *Object-Oriented Database Management Systems: Concepts and Issues.* Computer, April 1991. **24**(4): p. 33-47.

95. Harold, E.R. and W.S. Means, *XML in a Nutshell: a Desktop Quick Reference*. 1st edition ed. January 2001, Sebastopol, CA: O'Reilly & Associates, Inc.

96. Graves, M., *Designing XML databases*. 2002, Upper Saddle River, NJ 07458: Prentice Hall, Inc.

97. Peak, R., et al., *STEP, XML, and UML: Complementary Technologies.* Product Lifecycle Management (PLM) Special Issue. J. Computing & Information Science in Engineering, 2004. **4**(4): p. 379-390.

98. Ingalls, D., *Design Principles Behind Smalltalk.* Byte, 1981. **6**(8): p. 286.

99. *Nacubtisg MacApp 1.1.1 Programmer's Reference*. 1986, Cupertino, CA: Apple Computer.

100. Szyperski, C., *Component Software: Beyond Object-Oriented Programming*. 1997: Addison Wesley Longman Limited.

101. Uschold, M., R. Jasper, and P. Clark. *Three Approaches of Knowledge Sharing: a Comparative Analysis*. in *The 12th Workshop on Knowledge Acquisition, Modeling, and Management (KAW '99)*. 1999.

102. Uschold, M., R. Jasper, and P. Clark. *Three Approaches for Knowledge Sharing: a Comparative Analysis*. in *The 12th Workshop on Knowledge Acquisition, Modeling, and Management (KAW '99)*. 1999.

103. Leach, P., M. Mealling, and R. Salz, *A Universally Unique Identifier (UUID) URN Namespace*, in *RFC4122*. July 2005, Network Working Group.

104. Fowler, M., *Analysis Patterns: Reusable Object Models*. Addison-Wesley Object-Technology Series. 1997, Reading, MA: Addison Wesley Longman, Inc.

105. Willcox, K. and S. Wakayama, *Simultaneous Optimization of a Multiple-Aircraft Family.* Journal of Aircraft, 2003. **40**(4): p. 616-622.

106. *MySQL Reference Manual*. 2005, MySQL.com.

107. *http://www.hibernate.org*. Oct. 2005.

108. *http://xdoclet.sourceforge.net/xdoclet/tags/hibernate-tags.html*. Oct. 2005.

109. *http://jakarta.apache.org/commons/cli/, the CLI Component of Apache Jakarta Commons.* 2005.

110. *http://xerces.apache.org/, the website of Xerces.* Oct. 2005.

111. Bauer, C. and G. King, *Hibernate in Action*. 2005, Greenwich, CT: Manning Publications Co.

112. *http://java.sun.com/products/jdo/.* Oct. 2005.

113. *http://www.oracle.com/technology/products/ias/toplink/index.html.* Oct. 2005.

114. Betz, P., *Hibernate and XDoclet: How to Generate Hibernate Mapping Files with XDoclet*. 2004, http://www.downside.ch/hibernate/.

115. Trowbridge, D., et al., *Enterprise Solution Patterns Using Microsoft .NET: Version 1.0*. 2003: Microsoft Press.

116. Daum, B. and U. Merten, *System Architecture with XML*. 1st ed. The Morgan Kaufmann Series in Software Engineering and Programming. 2003, San Francisco, CA: Morgan Kaufmann Publishers.

117. *http://www.altova.com/products_ide.html.* Oct. 2005.

118. *http://www.mae.ncsu.edu/research/flight_research/f18/about.html.* Oct. 2005.

119. Guynn, M.D. and E.D. Olson, *Evaluation of an Aircraft Concept With Over-Wing, Hydrogen-Fueled Engines for Reduced Noise and Emissions*. 2002, NASA Langley Research Center: Hampton, Virginia.

120. Taewoo, N., S. Danielle, and D. N. Mavris. *A Generalized Aircraft Sizing Method and Application to Electric Aircraft*. in *The 3rd International Energy Conversion Engineering Conference*. 2005. San Francisco, California.

121. Taewoo, N., S. Danielle, and D. N. Mavris. *A Non-Deterministic Aircraft Sizing Method under Probabilistic Design Constraints*. in *The 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. 2006. Newport, Rhode Island.

122. *http://www.cessna.com, the Homepage of the Cessna Aircraft Company.* Dec. 2005.

123. *http://www.textron.com, the Homepage of the Textron Inc.* Dec. 2005.

124. *Jane's All the World's Aircraft*. 2003/2004 ed, ed. P. Jackson. 2003: Jane's Information Group Inc.

125. *http://www.harrisburgjetcenter.com/hjc/aircraft_sales/aircraft*. Dec. 2005

    /stationair/2002_cessna_stationair.htm. Dec. 2005.

126. Lu, Z., et al. *An Object-Oriented Sizing Approach for Revolutionary Aerospace Concepts*. in *AIAA 5th Aviation, Integration, and Operation Conference (ATIO)*. 2005. Arlington, VA.

127. *http://www.ballard.com, the website of Ballard Power Systems.* Dec. 2005.

128. *Levels of Information Systems Interoperability (LISI)*. 1998, C4ISR Interoperability Work Group, Department of Defense: DC.

129. Clark, T. and T. Moon, *Interoperability for Joint and Coalition Operations.* Australian Defense Force Journal, 2001. **151**(November/December): p. 23-36.

130. Kan, S.H., *Metrics and Models in Software Quality Engineering*. 2 ed. 2002: Addison Wesley. 560.

131. Manthorpe-Jr., W., *The Emerging Joint System of Systems: A Systems Engineering Challenge and Opportunity for APL.* John Hopkins APL Technical Digest, 1996. **17**(3): p. 305-310.

132. Liebowitz, J., *Knowledge Management: Learning from Knowledge Engineering*. 2001, Boca Raton, Florida: CRC Press.

133. Miller, E., *An Introduction to the Resource Description Framework*, in *D-Lib Magazine*. 1998.

134. *http://www.w3.org/RDF. W3C Official RDF Pages.* Oct. 2005.

135. *Hibernate: Hibernate Reference Documentation (Version: 3.0.5)*. Oct. 2005

136. *http://ant.apache.org, the Apache Ant Project.* Oct. 2005.

137. *http://www.w3.org/DOM/, the W3C DOM Standard.* Oct. 2005.

138. *http://www.saxproject.org/, the website of SAX.* Oct. 2005.

139. *http://www.mysql.com, MySQL and MySQL AB.* Oct. 2005.